Test Document

for

Automated Lecture Hall Booking Portal

Version 1.0

Prepared by

Group 12 **Group Name: Aviators**

Name	Roll no	Email
Chaitanya Bramhapurikar	230305	bcvishwas23@iitk.ac.in
Rahul Meena	230832	rahulkcm23@iitk.ac.in
Bhavya Chauhan	230294	bhavyach23@iitk.ac.in
Atharv Phirke	230238	atharvp23@iitk.ac.in
Aaradhya Rohi	230011	aaradhya23@iitk.ac.in
Chaudhari Divyesh	230325	divyesh23@iitk.ac.in
Devansh A Dhok	230354	devanshad23@iitk.ac.in
Areen Mahich	230188	areenm23@iitk.ac.in
Daksh Dua	220321	dakshdua22@iitk.ac.in
Avantika Rohite	230245	avantikar23@iitk.ac.in

Course: CS253

Mentor TA: Souvik Mukherjee

Date: 6 April 2025

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Aviators	The First version of the Test Document	06/04/2025

1 Introduction

Testing Strategy: System tests are automated whereas Integration Tests and Unit tests are implemented manually.

Testing Timeline: The testing was conducted after the implementation was completed.

Testers: The testing process was done by the developers themselves.

Coverage Criteria: We used decision coverage criteria and branch coverage criteria for unit testing.

2 Unit Testing

1. Email Thread

Unit Details: Class → EmailThread, Function → run

Test Owner: Bhavya Chauhan and Chaudhari Divyesh

Test Date: [27/03/2025] - [27/03/2025]

Test Results: Passed. Email was sent successfully in the background thread. The send_mail function was called with correct parameters, and thread execution completed without blocking main flow.

Inputs:

subject = "Test Subject"

message = "This is a test message."

from_email = "admin@example.com"

recipient_list = ["user@example.com"]

Output: Email sent in background without interrupting the main execution. [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

```
class EmailThread(threading.Thread):
    def __init__(self, subject, message, from_email, recipient_list):
        self.subject = subject
        self.message = message
        self.from_email = from_email
        self.recipient_list = recipient_list
        super().__init__()

    def run(self):
        send_mail(
            subject=self.subject,
            message=self.message,
            from_email=self.from_email,
            recipient_list=self.recipient_list,
            fail_silently=False,
        )
```

2. OTP Sending

Unit Details: Class → `SendOTP`, Function → `post`

Test Owner: Avantika Rohite and Aaradhya Rohi

Test Date:[27/03/2025] - [27/03/2025]

Test Results: Passed. OTP was generated, saved to user model, and sent via email in a background thread. Email composition and permissions were handled correctly.

Inputs: Authenticated user sends POST request to send OTP.

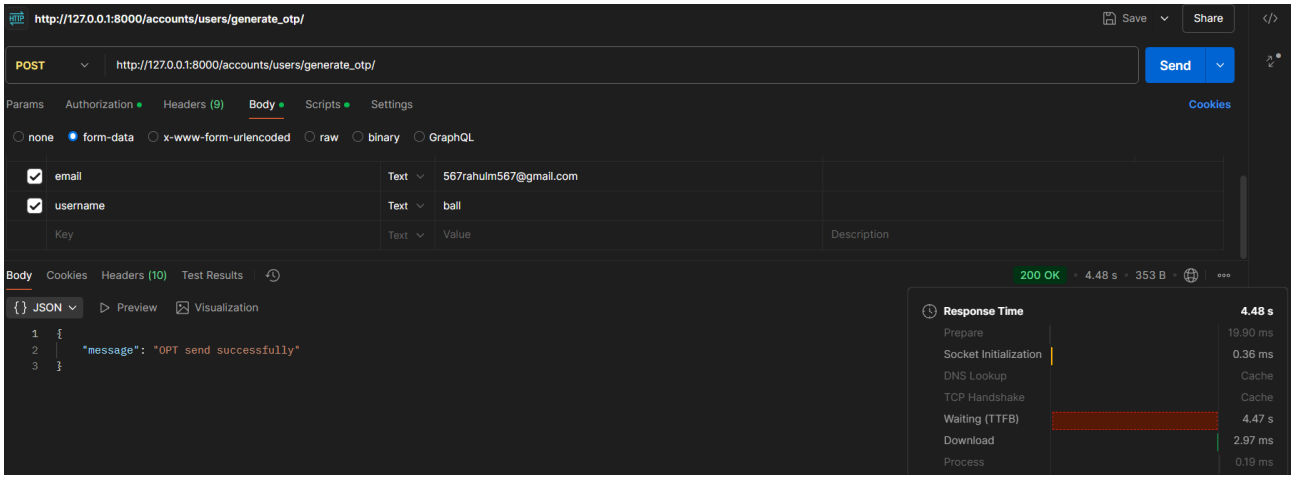
Output:

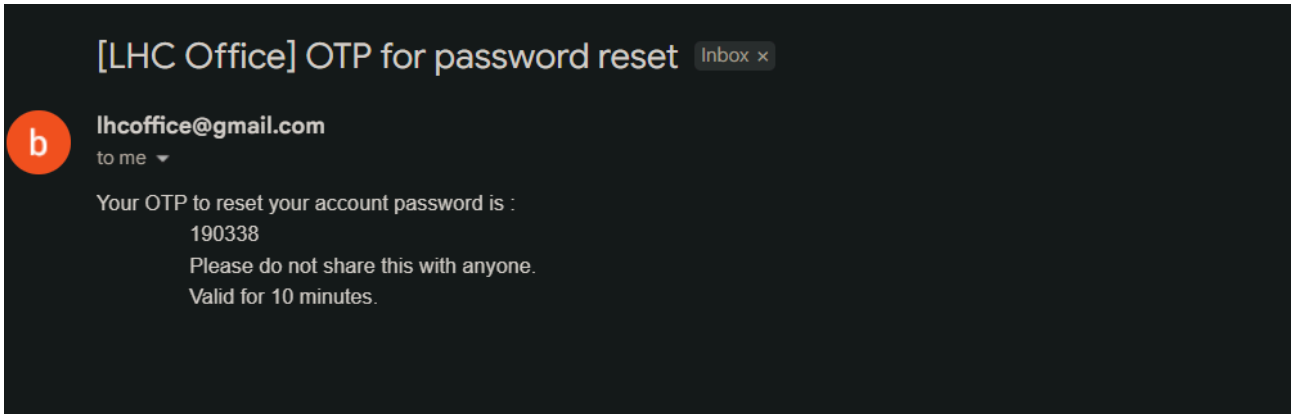
```
{
  "message": "OTP sent successfully"
}
```

OTP saved to user, email sent in background. [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

```python
class SendOTP(APIView):
    permission_classes = [Issameuser]

    def post(self, request, *args, **kwargs):
        user = request.user
        otp_code = str(random.randint(100000, 999999))

        # Update user model with new OTP
        user.otp = otp_code
        user.otp_created_at = now()
        user.save()

        # Compose email
        subject = '[LHC Office] OTP for password reset'
        message = f"""
Your OTP to reset your account password is:
{otp_code}
Please do not share this with anyone.
Valid for 10 minutes.
"""

        # Send email in background
        send_email_in_background(
            subject,
            message,
            settings.DEFAULT_FROM_EMAIL,
            [user.email],
        )

        return Response({"message": "OTP sent successfully"}, status=status.HTTP_200_OK)
```





3. OTP Request for Password Reset

Unit Details: Function → `request_reset_otp`

Test Owner: Devansh A Dhok and Areen Mahich

Test Date:[27/03/2025] - [27/03/2025]

Test Results: Passed. OTP was successfully generated, hashed, saved, and emailed when valid username and email matched. Errors correctly raised for mismatched or missing inputs.

Inputs:

```
{ "email": "user@example.com", "username": "user123" }
```

Output:

```
{ "message": "OTP sent successfully", "user_id": 5 }
```
[PASSED]

Inputs:

```
{ "email": "user@example.com", "username": "wronguser" }
```

Output:

```
{ "error": "User and registered email do not match" }
```
[PASSED]

Inputs:

Any field missing

Output:

`{ "error": "Email and username are required" }` [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

```python
@api_view(['POST'])
@permission_classes([AllowAny])
def request_reset_otp(request):
    email = request.data.get('email')
    username = request.data.get('username')

    if not email or not username:
        return Response({'error': 'Email and username are required'}, status=400)

    try:
        user = User.objects.get(email=email)
        user2 = User.objects.get(username=username)
        if user != user2:
            return Response({'error': 'User and registered email do not match'}, status=404)
        otp_code = str(random.randint(100000, 999999))
        hashed_otp = make_password(otp_code)
        OTPReset.objects.create(user=user, otp=hashed_otp)

        send_email_in_background(
    '[LHC Office] OTP for password reset',
    f"""Your OTP to reset your account password is :
    {otp_code}
    Please do not share this with anyone.
    Valid for 10 minutes.
    """,
    settings.DEFAULT_FROM_EMAIL,
    [user.email],
)
        return Response({'message': 'OTP sent successfully', 'user_id': user.id})
    except User.DoesNotExist:
        return Response({'error': 'User not found'}, status=404)
```

4. OTP Verification and Password Reset

Unit Details: Function → `verify_and_reset_password`

Test Owner: Chaitanya Bramhapurikar and Daksh Dua

Test Date:[27/03/2025] - [27/03/2025]

Test Results: Passed. Password was successfully updated when a valid and unexpired OTP was provided. Errors raised appropriately for invalid/expired OTPs, missing fields, or non-existent user.

Inputs:

```
{
  "user_id": 5,
  "otp": "123456",
  "new_password": "new_secure_pass"
}
```

Output:

`{ "message": "Password reset successful" }` [PASSED]

Inputs:

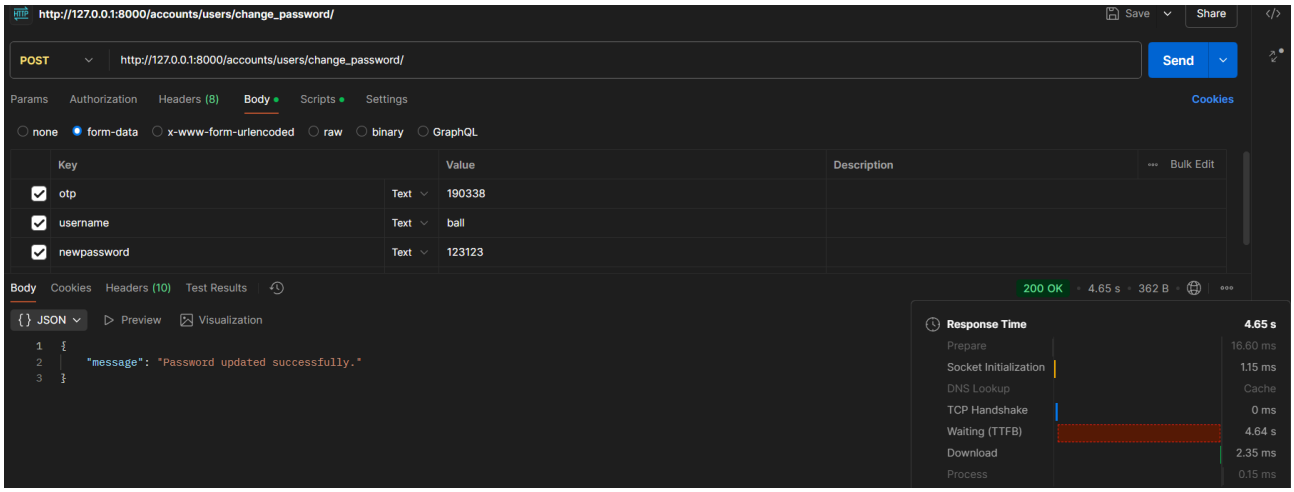
Expired or wrong OTP

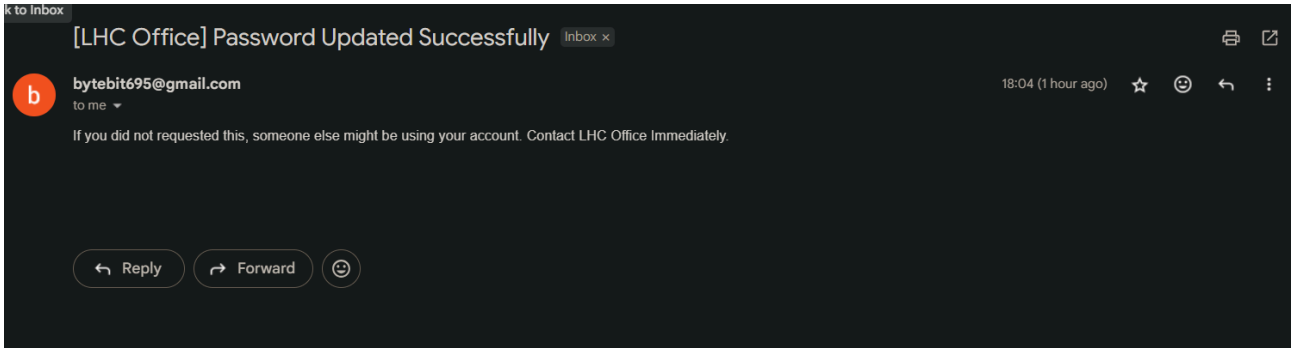
Output:

`{ "error": "Invalid or expired OTP" }` [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

```python
@api_view(['POST'])
@permission_classes([AllowAny])
def request_reset_otp(request):
    email = request.data.get('email')
    username = request.data.get('username')

    if not email or not username:
        return Response({'error': 'Email and username are required'}, status=400)

    try:
        user = User.objects.get(email=email)
        user2 = User.objects.get(username=username)
        if user != user2:
            return Response({'error': 'User and registered email do not match'}, status=404)
        otp_code = str(random.randint(100000, 999999))
        hashed_otp = make_password(otp_code)
        OTPReset.objects.create(user=user, otp=hashed_otp)

        send_email_in_background(
    '[LHC Office] OTP for password reset',
    f"""Your OTP to reset your account password is :
    {otp_code}
    Please do not share this with anyone.
    Valid for 10 minutes.
    """,
    settings.DEFAULT_FROM_EMAIL,
    [user.email],
)
        return Response({'message': 'OTP sent successfully', 'user_id': user.id})
    except User.DoesNotExist:
        return Response({'error': 'User not found'}, status=404)
```





5. Reject Booking

Unit Details: Function → `reject_booking`

Test Owner: Rahul Meena and Atharv Phirke

Test Date:[27/03/2025] - [27/03/2025]]

Test Results: Passed. Bookings were successfully rejected when valid tokens were provided and the booking was still pending. Proper error responses returned for invalid, expired, or missing tokens. Confirmation email sent on successful rejection.

Inputs:

Query Params: `?booking_token=abc123&authority_token=xyz456`

Output:

Renders `Rejected.html` and deletes the booking. [PASSED]

Inputs:

Missing `booking_token` or `authority_token`

Output:

`{ "error": "Invalid or missing rejection token." }` [PASSED]

Inputs:

Booking token expired or booking already processed

Output:

`{ "error": "Token expired or invalid" }` [PASSED]

Inputs:

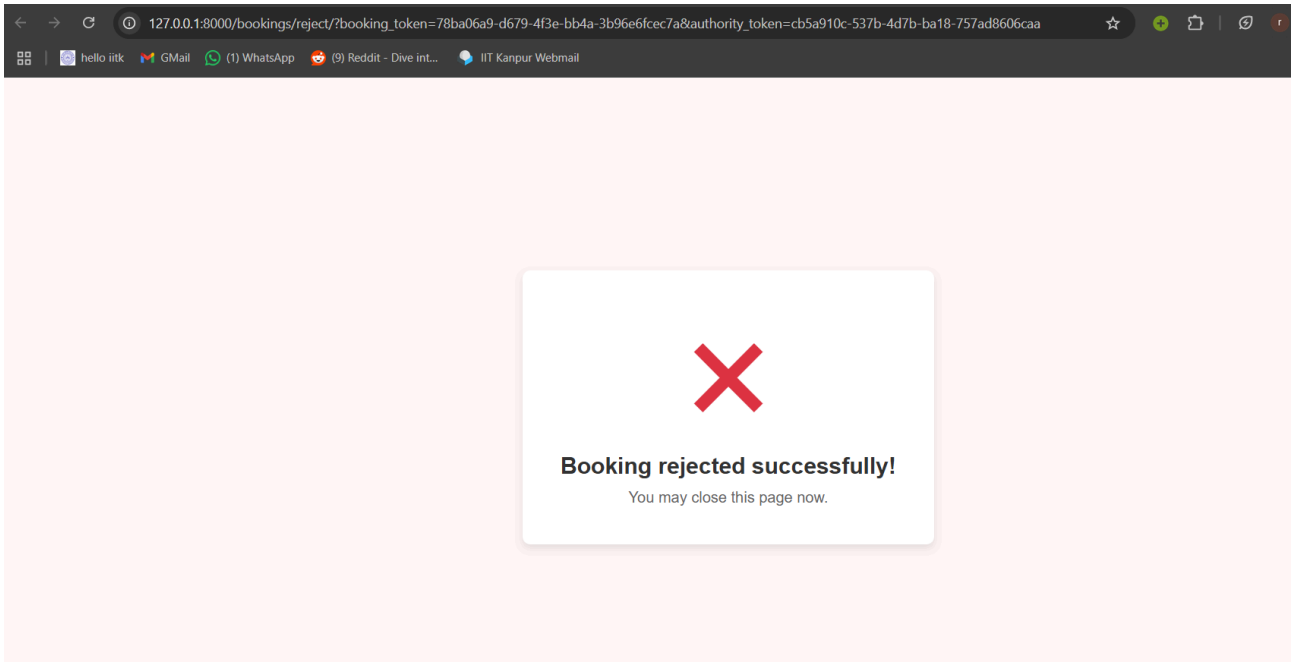
Booking already approved/rejected

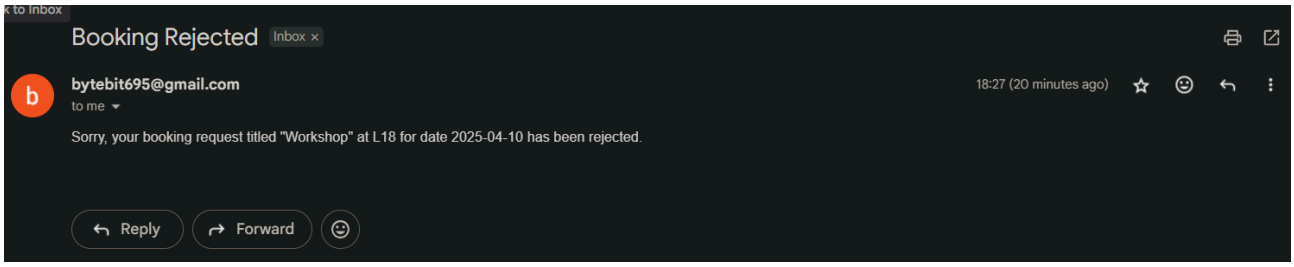
Output:

Renders `Booking Processed.html` with booking status. [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

```
def reject_booking(request):
    booking_token = request.GET.get("booking_token")
    authority_token = request.GET.get("authority_token")

    if not booking_token or not authority_token:
        return JsonResponse({"error" : "Invalid or missing rejection token."}, status=400)

    booking = get_object_or_404(Booking, booking_token=booking_token)
    if booking.status != 'pending': return JsonResponse({"error": "Token expired or invalid"}, status=400)
    if not booking or booking.token_expiry < timezone.now():return JsonResponse({"error": "Token expired or invalid"}, status=400)

    authority_name = next(
        (name for name, token in booking.authority_tokens.items() if token == authority_token),
        None
    )

    if booking.status in ["Approved", "Rejected"]:
        return render(request, "bookings/Booking Processed.html", context={"status":booking.status}, status=200)

    booking.status = "Rejected"
    booking.approvals_pending = {}
    booking.save()

    send_email_in_background(
    'Booking Rejected',
    f'Sorry, your booking request titled "{booking.title}" at {booking.room.name} for date {booking.booking_date} has been rejected.',
    settings.DEFAULT_FROM_EMAIL,
    [booking.creator.email],
)

    booking.delete()
    return render(request, "bookings/Rejected.html", context={}, status=200);
```





6. Approve Booking

Unit Details: Function → `approve_booking`

Test Owner: Chaudhari Divyesh and Daksh Dua

Test Date:[27/03/2025] - [27/03/2025]

Test Results: Passed. Under valid conditions, the booking was approved, prior pending requests for the same slot were rejected, and appropriate confirmation/notification emails were sent. All invalid or expired token paths handled properly.

Inputs:

Query Params: `?booking_token=abc123&authority_token=auth456`

Output:

Renders `Confirmed.html` when all approvals are complete and updates booking status to "Approved". [PASSED]

Inputs:

Booking already approved or rejected

Output:

Renders `Booking Processed.html` with status. [PASSED]

Inputs:

Missing or expired booking_token / authority_token

Output:

`{ "error": "Invalid or missing approval token." }` or `{ "error": "Token expired or invalid" }` [PASSED]

Inputs:

Booking not in `pending` state

Output:

`{ "error": "This booking is not available for approval" }` [PASSED]

Inputs:

Booking partially approved

Output:

Renders `Approved.html`, triggers email to next approver. [PASSED]

Inputs:

Duplicate approval from the same authority

Output:

Renders `AlreadyApproved.html`. [PASSED]

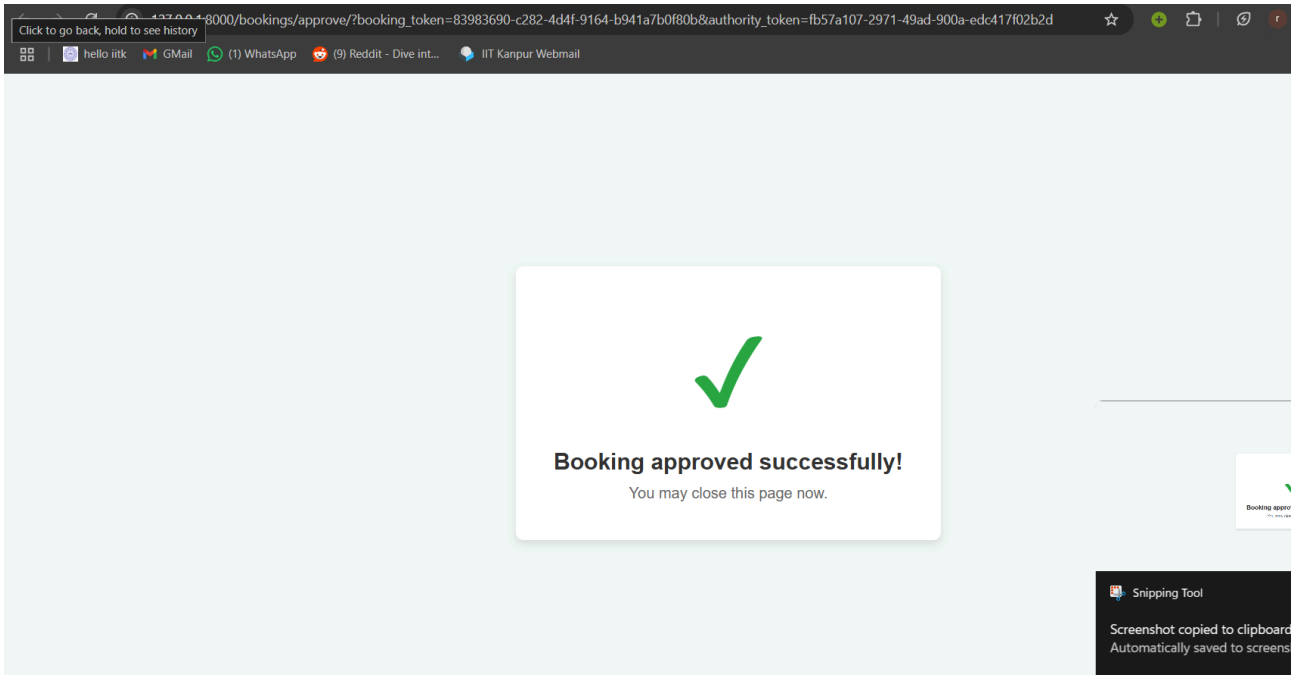
Inputs:

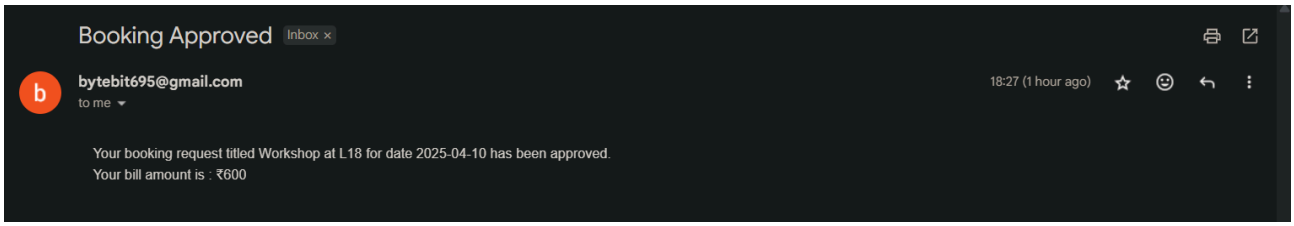
Pending bookings exist for the same slot

Output:

Rejects conflicting pending bookings, sends rejection emails. [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%





7. Room Search

Unit Details: Class → `RoomSearchView`, Method → `get_queryset`

Test Owner: Bhavya Chauhan and Areen Mahich

Test Date:[27/03/2025] - [27/03/2025]

Test Results: Passed. Filters, sorting, and search functionality all worked correctly. The capacity filter handled both valid and invalid input gracefully. Accessory filters correctly applied when set to `true`.

Inputs:

Query Params: `?capacity=80&has_ac=true&has_projector=true`

Output:

Returns rooms with capacity ≥ 80 and both AC and projector enabled. [PASSED]

Inputs:

Query Params: `?name=LT105`

Output:

Returns room(s) matching name "LT105". [PASSED]

Inputs:

Query Params: `?ordering=capacity`

Output:

Returns rooms sorted by capacity. [PASSED]

Inputs:

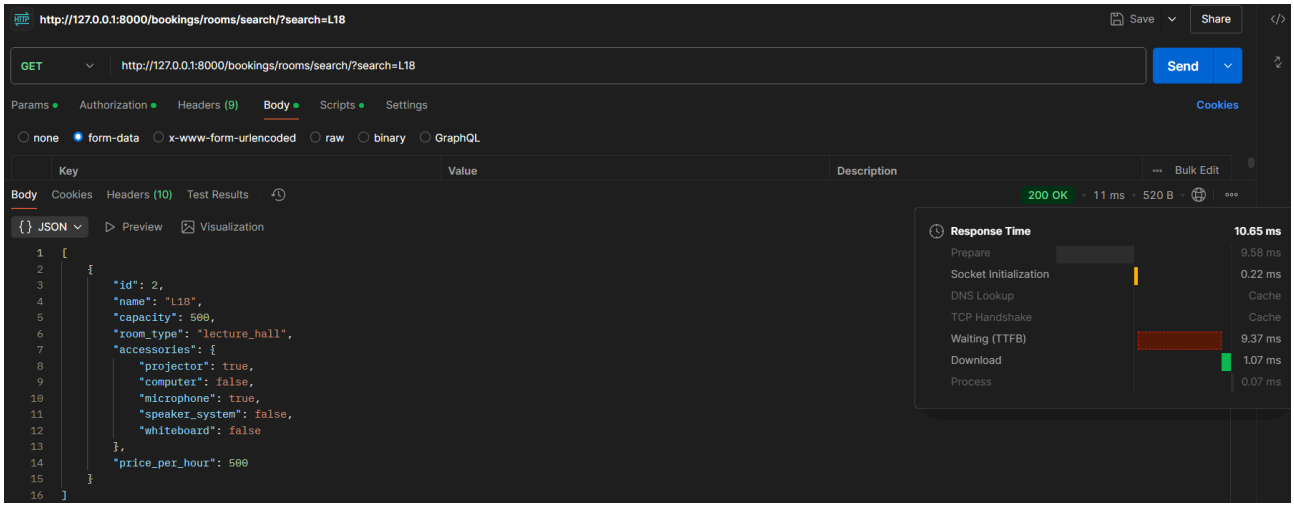
Query Params: `?capacity=invalid`

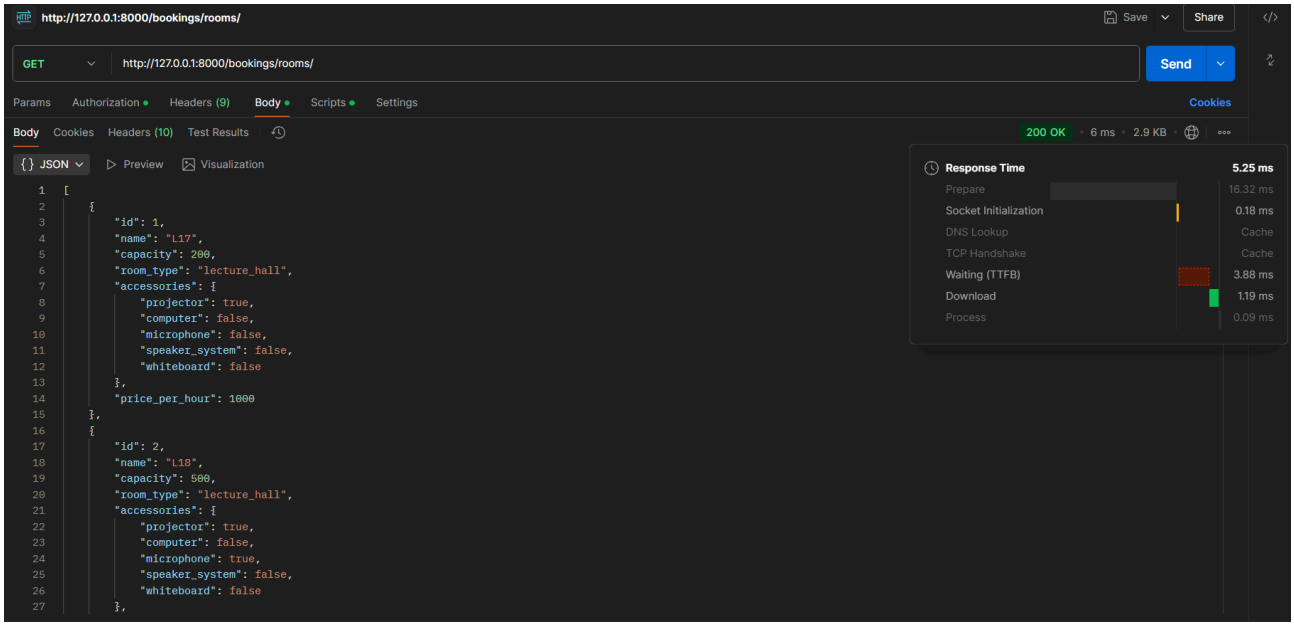
Output:

Returns full room list without filtering by capacity (gracefully handles bad input). [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

```python
class RoomSearchView(generics.ListAPIView):
    # queryset = Room.objects.all()
    serializer_class = RoomSerializer
    filter_backends = [DjangoFilterBackend, filters.SearchFilter, filters.OrderingFilter]

    # Exact field filters
    filterset_fields = [
        'room_type',  # Filter by room type (tutorial or lecture_hall)
        'has_ac',     # Filter by AC availability
        'has_board',  # Filter by board availability
        'has_projector',  # Filter by projector availability
        'price_per_hour',  # Filter by price per hour
    ]

    # Full-text search fields
    search_fields = [
        'name',  # Search by room name
    ]

    # Dynamic filtering for accessories
    def get_queryset(self):
        queryset = Room.objects.all()
        for accessory in dict(Room.ACCESSORY_OPTIONS):
            if self.request.query_params.get(accessory) == 'true':
                queryset = queryset.filter(**{f'accessories__{accessory}': True})
        return queryset

    # Sorting options
    ordering_fields = [
        'name',           # Sort by room name
        'capacity',       # Sort by capacity
        'price_per_hour', # Sort by price per hour
    ]
    ordering = ['name']  # Default sorting (alphabetical by room name)

    def get_queryset(self):
        queryset = Room.objects.all()
        capacity = self.request.query_params.get('capacity', None)
        if capacity is not None:
            try:
                capacity = int(capacity)
                queryset = queryset.filter(capacity__gte=capacity)
            except ValueError:
                pass  # Ignore invalid capacity values
        return queryset
```




http://127.0.0.1:8000/bookings/rooms/
Save
Share
GET
http://127.0.0.1:8000/bookings/rooms/
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
Body
Cookies
Headers (10)
Test Results
200 OK
6 ms
2.9 KB
JSON
Preview
Visualization
Response Time
5.25 ms
Prepare
16.32 ms
Socket Initialization
0.18 ms
DNS Lookup
Cache
TCP Handshake
Cache
Waiting (TTFB)
3.88 ms
Download
1.19 ms
Process
0.09 ms
[
{
"id": 1,
"name": "L17",
"capacity": 200,
"room_type": "lecture_hall",
"accessories": {
"projector": true,
"computer": false,
"microphone": false,
"speaker_system": false,
"whiteboard": false
},
"price_per_hour": 1000
},
{
"id": 2,
"name": "L18",
"capacity": 500,
"room_type": "lecture_hall",
"accessories": {
"projector": true,
"computer": false,
"microphone": true,
"speaker_system": false,
"whiteboard": false
},

8. Booking Search

Unit Details: Class → `BookingSearchView`, Method → `ListAPIView`

Test Owner: Devansh A Dhok and Chaitanya Bramhapurikar

Test Date:[27/03/2025] - [27/03/2025]

Test Results: Passed. All filters, search, and sorting fields responded as expected. Ensured appropriate bookings returned.

Inputs: `?status=pending&room=5`

Output: Returns all pending bookings for Room ID 5. [PASSED]

Inputs: `?search=LT101`

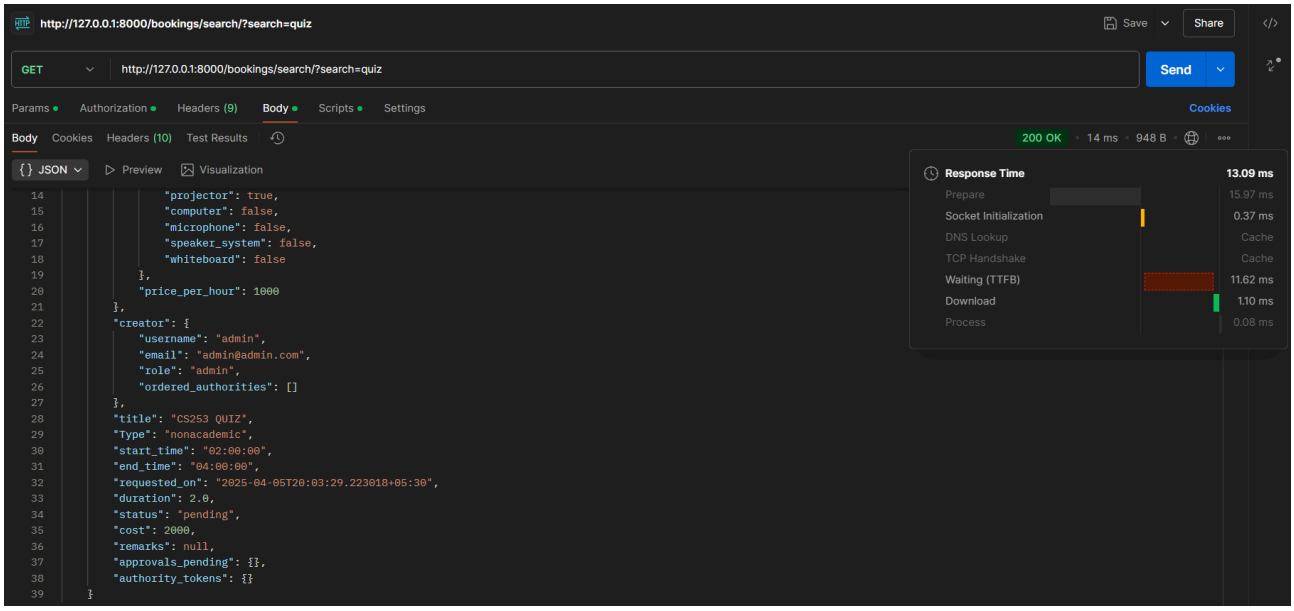
Output: Returns bookings with room name "LT101" or titles containing "LT101". [PASSED]

Inputs: `?ordering=requested_on`

Output: Returns bookings in ascending order of `requested_on`. [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

```python
class BookingSearchView(generics.ListAPIView):
    queryset = Booking.objects.all()
    serializer_class = BookingSerializer
    permission_classes = [AllowAny]
    filter_backends = [DjangoFilterBackend, filters.SearchFilter, filters.OrderingFilter]

    # Exact field filters
    filterset_fields = ['status', 'Type', 'room', 'creator', 'booking_date']

    # Full-text search fields
    search_fields = ['title', 'room__name', 'creator__username']

    # Sorting options
    ordering_fields = ['start_time', 'end_time', 'requested_on']
    ordering = ['-start_time']  # Default sorting (latest bookings first)
```



9. User Creation

Unit Details: Class → `UserListCreateView`, Function → `create()`

Test Owner: Avantika Rohite and Aaradhya Rohi

Test Date:[27/03/2025] - [27/03/2025]

Test Results: Passed. User created successfully with and without authority mapping. Proper validation errors raised on invalid input.

Inputs:

```
{
  "username": "test_admin",
  "email": "admin@example.com",
  "user_type": "admin"
}
```

Output: User created successfully. Status: 201 [PASSED]

Inputs:

```
{
  "username": "test_student",
  "email": "student@example.com",
  "user_type": "student",
  "authorities": [3, 1, 2]
}
```

Output: User created with authorities in correct order. Status: 201 [PASSED]

Inputs:

Missing required fields like email or password

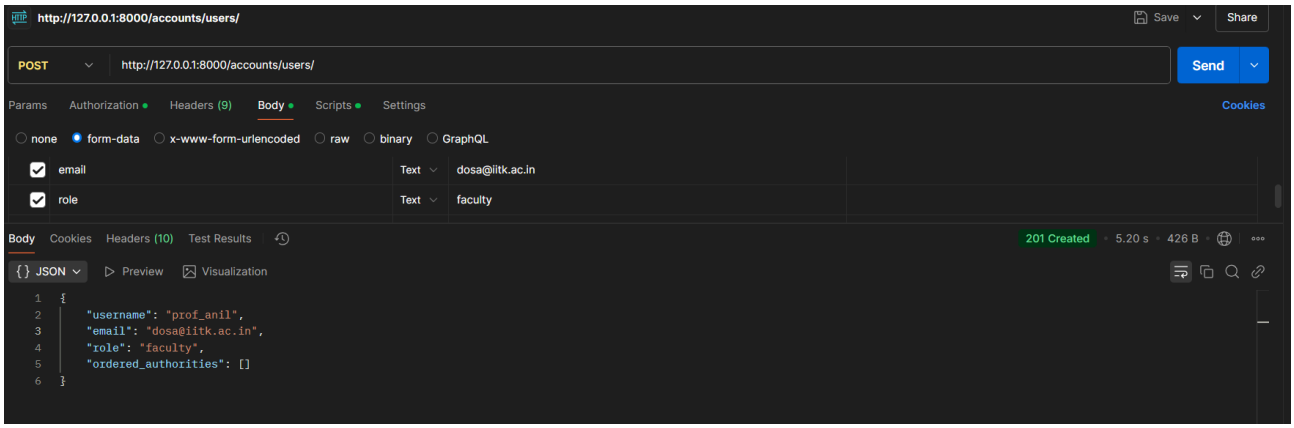
Output: Validation error raised. Status: 400 [PASSED]

Inputs:

Invalid `authorities` list (e.g., non-existent IDs)

Output: Validation error raised due to foreign key constraint. [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%



10. User Booking History

Unit Details: Class → `UserBookingHistoryView`, Method → `get_queryset()`

Test Owner: Atharv Phirke and Rahul Meena

Test Date:[29/03/2025] - [29/03/2025]

Test Results: Passed. Only authenticated users could access their own booking history.

Inputs: Authenticated user "ashwin" makes GET request

Output: Returns only bookings created by "ashwin", sorted by most recent request. [PASSED]

Inputs: Unauthenticated user tries to access

Output: Access denied (401 Unauthorized). [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

```python
class UserBookingHistoryView(generics.ListAPIView):
    serializer_class = BookingSerializer
    permission_classes = [IsAuthenticated]  # Only authenticated users can see their history

    def get_queryset(self):
        """Return only the bookings of the logged-in user"""
        return Booking.objects.filter(creator=self.request.user).order_by('-requested_on')
```

11. Send Rejection Mail

Unit Details: Function → `send_rejection_mail(request, booking_id)`

Test Owner: Rahul Meena and Chaudhari Divyesh

Test Date:[29/03/2025] - [29/03/2025]

Test Results: Passed. Rejection emails sent and bookings deleted as expected under valid inputs. Proper exceptions raised on bad input or method.

Inputs:

`Method: POST, Body: {"remark": "Not sufficient justification"}, booking_id=5`

Output: Booking rejected, email sent, booking deleted. Status: 204 No Content. [PASSED]

Inputs:

`Method: GET, booking_id=5`

Output: Returns error "Invalid request method". Status: 405. [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

```python
@csrf_exempt
def send_rejection_mail(request, booking_id):
    if request.method != "POST":
        return JsonResponse({"error": "Invalid request method"}, status=405)

    try:
        data = json.loads(request.body)
        remarks = data.get("remark", "")

        if not remarks:
            return JsonResponse({"error": "Remarks are required"}, status=400)

        # Get the booking object
        booking = get_object_or_404(Booking, id=booking_id)
        booking.status = 'rejected'

        # Compose the email content
        subject = 'Booking Rejected'
        message = (
            f'Sorry, your booking request titled "{booking.title}" at {booking.room.name} '
            f'for date {booking.booking_date} has been rejected by LECTURE HALL ADMINISTRATION.\n'
            f'Remarks: {remarks}'
        )
        from_email = settings.DEFAULT_FROM_EMAIL
        recipient_list = [booking.creator.email]

        # Send the email in the background
        send_email_in_background(subject, message, from_email, recipient_list)

        # Delete the booking
        booking.delete()
        return JsonResponse({"message": "Deleted successfully"}, status=204)
```

12. BookingCRUDView

Unit Details: Class-based view → `BookingCRUDView`

Test Owner: Aaradhya Rohi and Avantika Rohite

Test Date:[29/03/2025] - [29/03/2025]

Test Results: All test cases passed including edge conditions. Booking creation, retrieval, updating, and deletion were verified for different user roles.

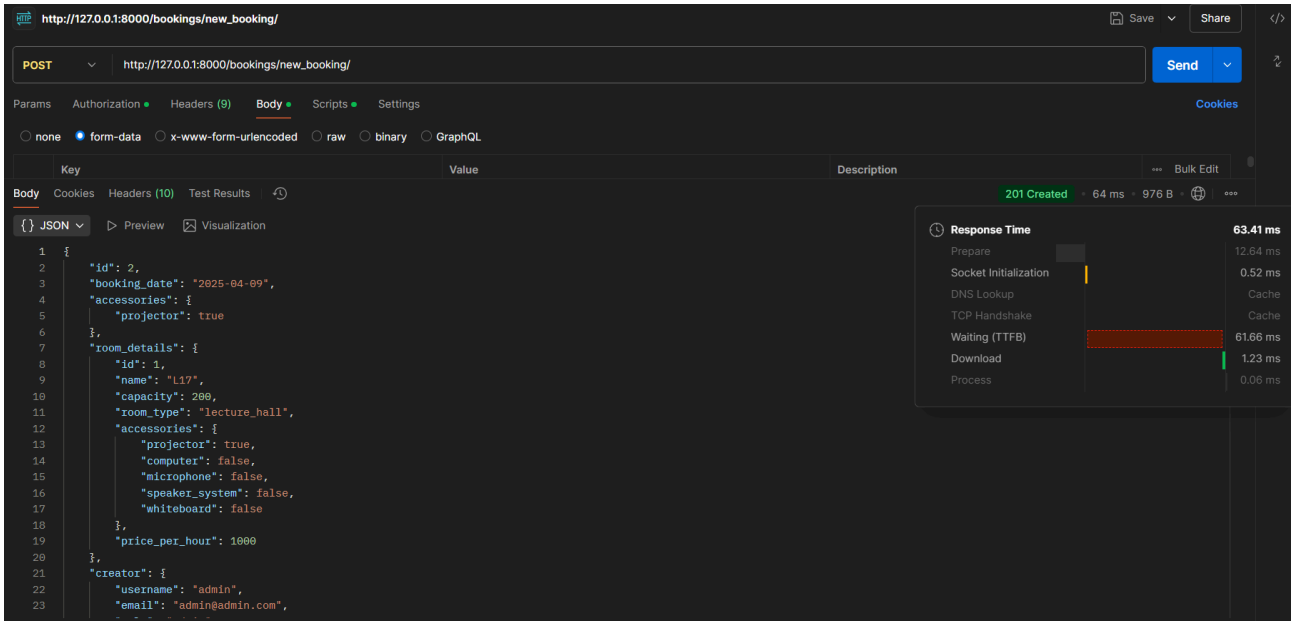
- **POST – Booking Creation**

Inputs:

Logged-in `user`: `admin`,

Payload: Valid booking data with available accessories

Output: Booking created successfully with status “pending”, email sent to first authority [PASSED]

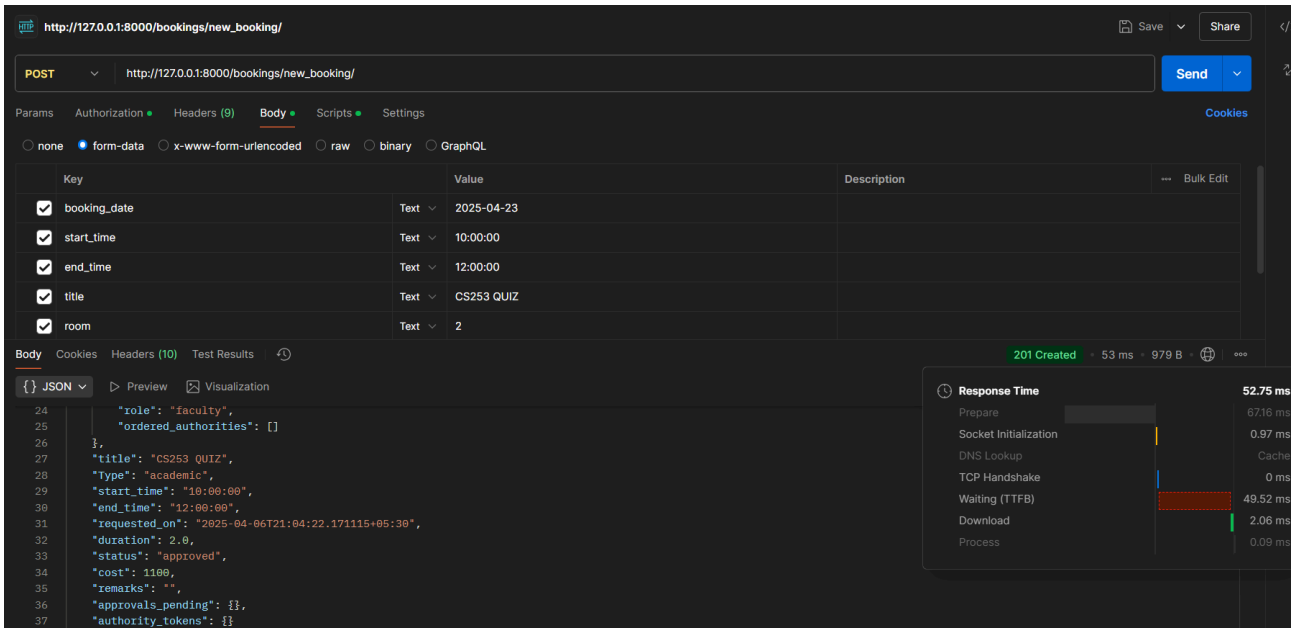


Inputs:

Logged-in `user`: `Prof_Saha`,

Payload: Valid academic booking data

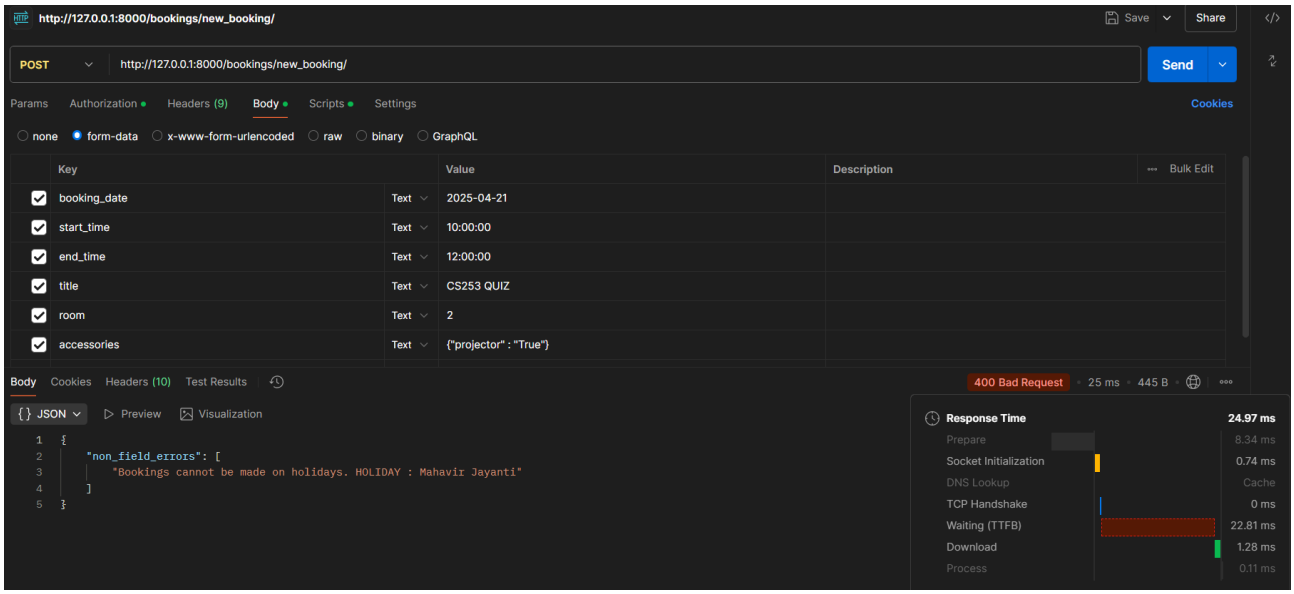
Output: Booking created with status “approved”, no authority email sent [PASSED]



Inputs:

Invalid `booking_date` included , Holiday Date. (2025-04-21 Mahavir Jayanti)

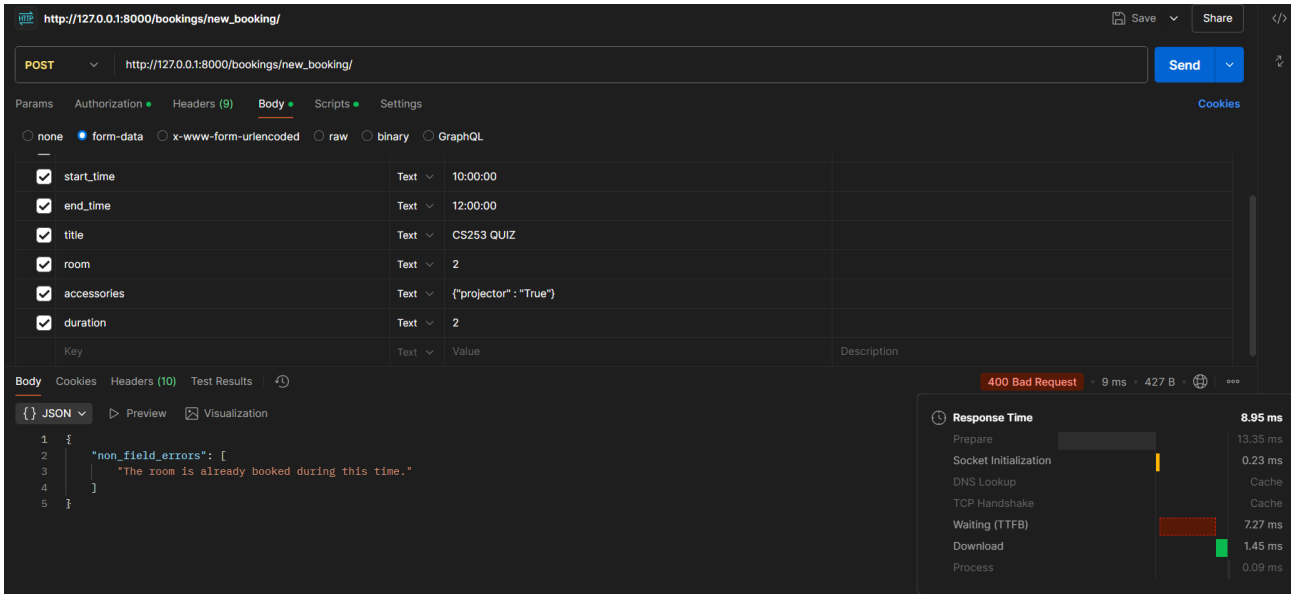
Output: `ValidationError` raised, error 400 Bad request returned:



Inputs:

Invalid `room` included , Room Already Booked on that date and time.

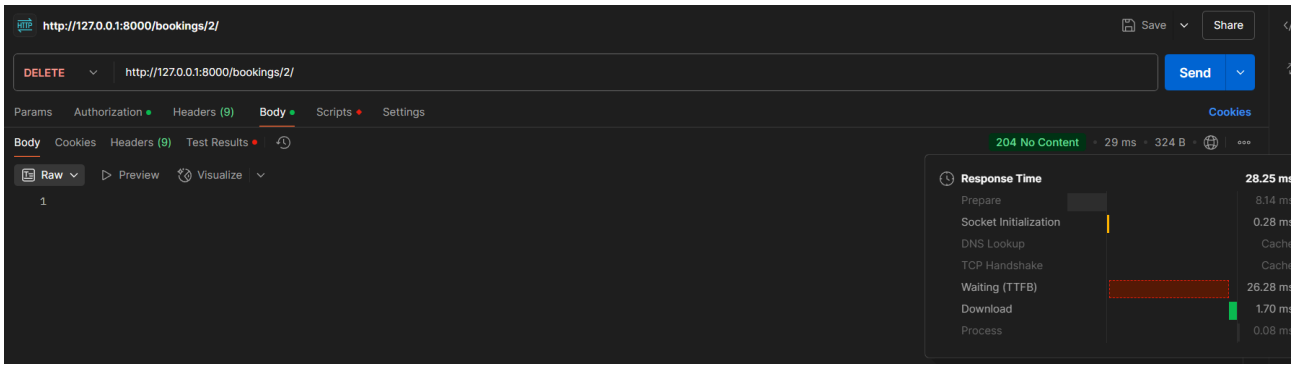
Output: `ValidationError` raised, error 400 bad request returned:



- **DELETE – Booking Deletion**

Inputs: Booking id to be deleted included in api endpoint.

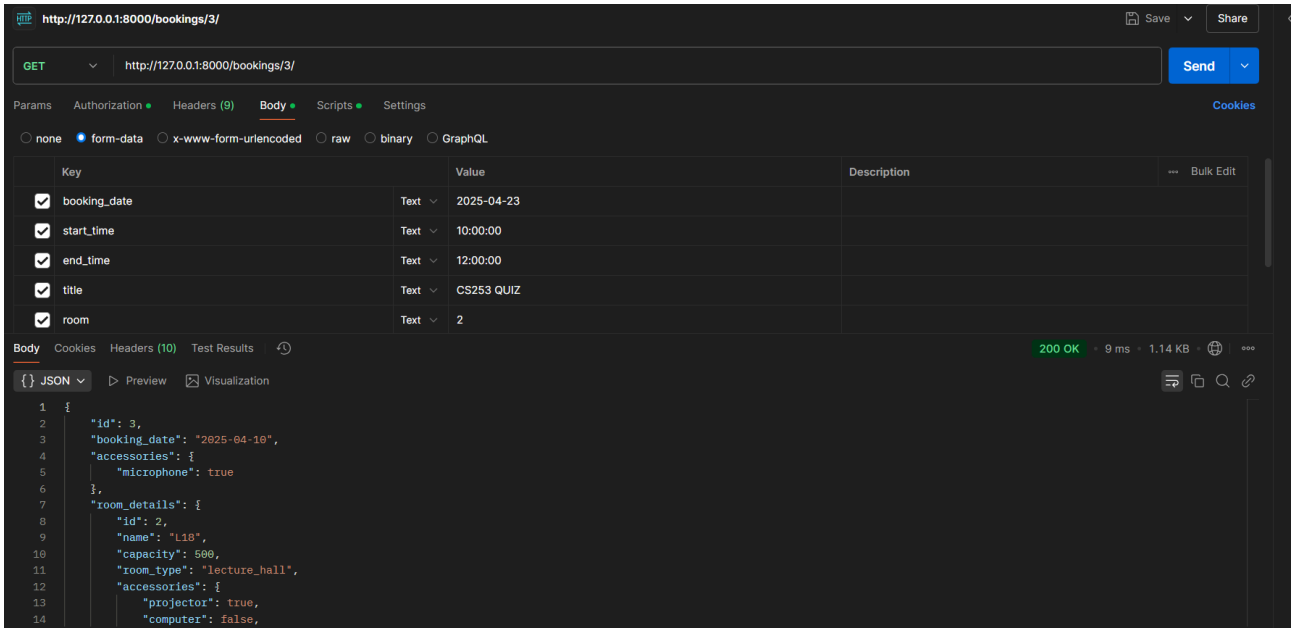
Output: `Success, 204 No content returned`



- **GET - Booking retrieve**

Inputs: Booking id to be retrieved included in URL.

Output: `200 OK status,` Booking with id returned



Available Booking Slots

Unit Details: Class → `AvailableBookingSlotsView`, Function → `post()`

Test Owner: Bhavya Chauhan and Chaitanya Bramhapurikar

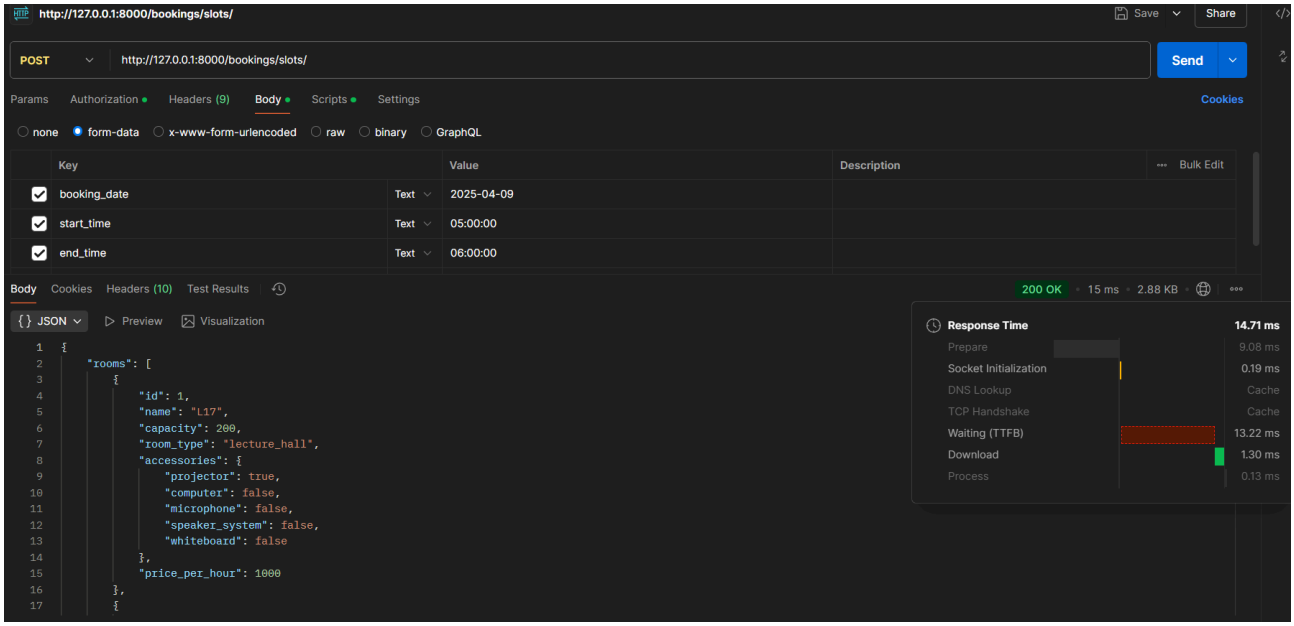
Test Date: [01/04/2025] - [01/04/2025]

Test Results: Passed. Returns appropriate rooms when available. Proper validation and error messages for invalid inputs.

Inputs:

```
{
  "booking_date": "2025-04-02",
  "start_time": "10:00:00",
  "end_time": "12:00:00"
}
```

Output: List of available rooms. Status: 200 OK. [PASSED]



Inputs:

```
{
  "booking_date": "2025-04-02",
  "start_time": "12:00:00"
}
```

Output: Error → "booking_date, start_time, and end_time are required." Status: 400. [PASSED]

Inputs:

```
{
  "booking_date": "2025-04-02",
  "start_time": "15:00:00",
  "end_time": "14:00:00"
}
```

Output: Error → "end_time must be later than start_time." Status: 400. [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

13. Feedback Handling

Unit Details: Class → CreateFeedbackView, Functions → post, get

Test Owner: Avantika and Aaradhya Rohi

Test Date:[29/03/2025] - [29/03/2025]

Test Results: Passed. Authenticated users are able to post and retrieve feedback successfully. Feedback is stored only when valid data is provided.

Inputs:

rating = "5"

comment = "Excellent experience!"

Output: Feedback submitted successfully [PASSED]

Inputs:

rating = ""

comment = "Good"

Any one field empty

Output: Feedback submission failed [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

```python
class CreateFeedbackView(APIView):
    permission_classes = [permissions.IsAuthenticated]

    def post(self, request):
        # print(request.data.get('rating'))
        serializer = FeedbackSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save(user=request.user)
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
    def get(self, request):
        feedbacks = Feedback.objects.all().order_by('-created_at')  # newest first
        serializer = FeedbackSerializer(feedbacks, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)
```

14. Send Approval Email

Unit Details: Function → `send_approval_email`

Test Owner: Areen Mahich and Daksh Dua

Test Date:[29/03/2025] - [29/03/2025]

Test Results: Passed. Email sent successfully to the authority when token was valid. Function exited gracefully when token was missing.

Inputs:

`authority_email`: `"authority1@iitk.ac.in"`

`booking`: Valid booking object with all fields populated and matching authority token

Output: Email sent successfully to authority with correct approval and rejection links [PASSED]

Inputs:

`authority_email`: `"invalid@iitk.ac.in"` (not present in `authority_tokens`)

`booking`: Valid booking object

Output: No email sent, function exited safely without error [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

15. Download Bill

Unit Details: Class → `DownloadBillPDF`, Function → `get`

Test Owner: Bhavya Chauhan and Devansh A Dhok

Test Date:[29/03/2025] - [29/03/2025]

Test Results: Passed. PDF was successfully generated and returned in the HTTP response.

The `generate_bill` function was invoked with correct booking data.

Permission checks using `DownloadPermissions` were enforced properly.

The response contained accurate headers (`Content-Type`, `Content-Disposition`) and valid PDF content.

The frontend received the response as a **Blob object** with MIME type `"application/pdf"`.

This Blob was then converted into a downloadable link and triggered for download using:14.

Inputs:

`booking_id = 101`

Authenticated user with permission

(Booking with ID 101 exists and belongs to the user)

Output: PDF sent as downloadable response with status 200. Headers `Content-Type` and `Content-Disposition` were correct. PDF content included booking reference, event name, date, time, hall, user, duration, and charges. [PASSED]

Structural Coverage: Code Coverage: 100%, Branch Coverage: 100%

16. Generatebill

Unit Details: Class → Standalone Function, Function → `generate_bill`

Test Owner: Areen Mahich and Chaitanya Bramhapurikar

Test Date:[29/03/2025] - [29/03/2025]

Test Results:

Passed. PDF was successfully generated using reportlab and returned as a non-empty `BytesIO` object.

The output buffer was validated as a proper PDF. Core data such as booking reference, event name, hall, date, time, user, duration, and charges were found in the generated content.

PDF integrity was verified using PyPDF2, and expected text fields were successfully parsed.

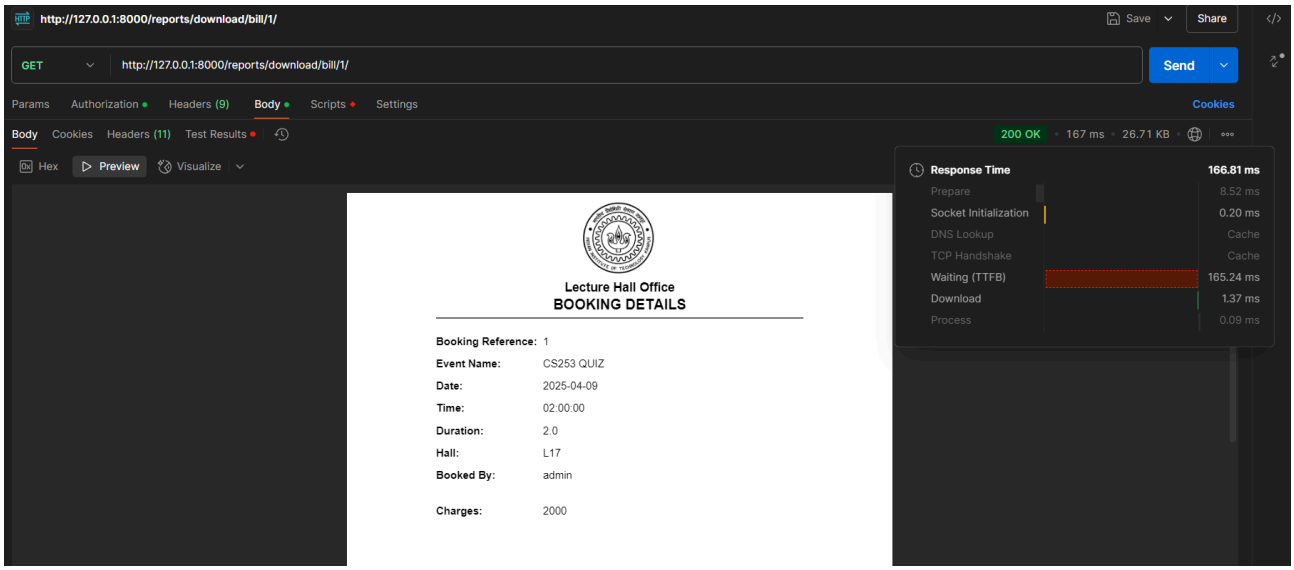
Inputs:

- ```
  data = {
      "booking_ref": "123",
      "event_name": "Seminar on Robotics",
      "date": "2025-04-01",
      "time": "10:00 AM",
      "duration": "2 hours",
      "hall_name": "LHC-101",
      "booked_by": "bhavya",
      "charges": "₹1500"
  }
  ```
  ```

Output:

- Returned a `BytesIO` object with valid PDF content.
- PDF contained all expected booking fields.
- File is ready for Blob-type frontend consumption and is converted to and downloaded as a `.pdf` file.
- Verified presence of key strings: `"Booking Reference:"`, `"Seminar on Robotics"`, `"LHC-101"`
- [**PASSED**]
- **Structural Coverage:** Code Coverage: 100%, Branch Coverage: 100%

```python
class DownloadBillPDF(APIView):
    queryset = Booking.objects.all()
    permission_classes = [DownloadPermissions]
    def get(self, request, booking_id):
        # Get booking data (replace with your actual query)
        booking = get_object_or_404(Booking, id=booking_id)
        self.check_object_permissions(request, booking)

        # Create PDF buffer
        data = {
            "booking_ref" : str(booking.id),
            "event_name": booking.title,
            "date": str(booking.booking_date),
            "time": str(booking.start_time),
            "hall_name": booking.room.name,
            "booked_by": booking.creator.username,
            "charges": str(booking.cost),
            "duration" : str(booking.duration),
            "remarks" : (booking.remarks)
        }
        buffer = generate_bill(data)
        response = HttpResponse(buffer, content_type='application/pdf')
        response['Content-Disposition'] = f'attachment; filename="bill_booking_{booking.id}.pdf"'
        return response
# Create your models here.
```


http://127.0.0.1:8000/reports/download/bill/1/
Save
Share
GET
http://127.0.0.1:8000/reports/download/bill/1/
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
Body
Cookies
Headers (11)
Test Results
200 OK
167 ms
26.71 KB
Hex
Preview
Visualize
Lecture Hall Office
BOOKING DETAILS
Booking Reference: 1
Event Name: CS253 QUIZ
Date: 2025-04-09
Time: 02:00:00
Duration: 2.0
Hall: L17
Booked By: admin
Charges: 2000
Response Time
166.81 ms
Prepare
8.52 ms
Socket Initialization
0.20 ms
DNS Lookup
Cache
TCP Handshake
Cache
Waiting (TTFB)
165.24 ms
Download
1.37 ms
Process
0.09 ms

3 Integration Testing

1. Login

Module Details: In this we tested the different login (admin and non admin) through various api calls including 'login' as well as the frontend for dashboard based on different logins.

Test Owner: Areen and Daksh

Test Date:[29/03/2025] - [29/03/2025]

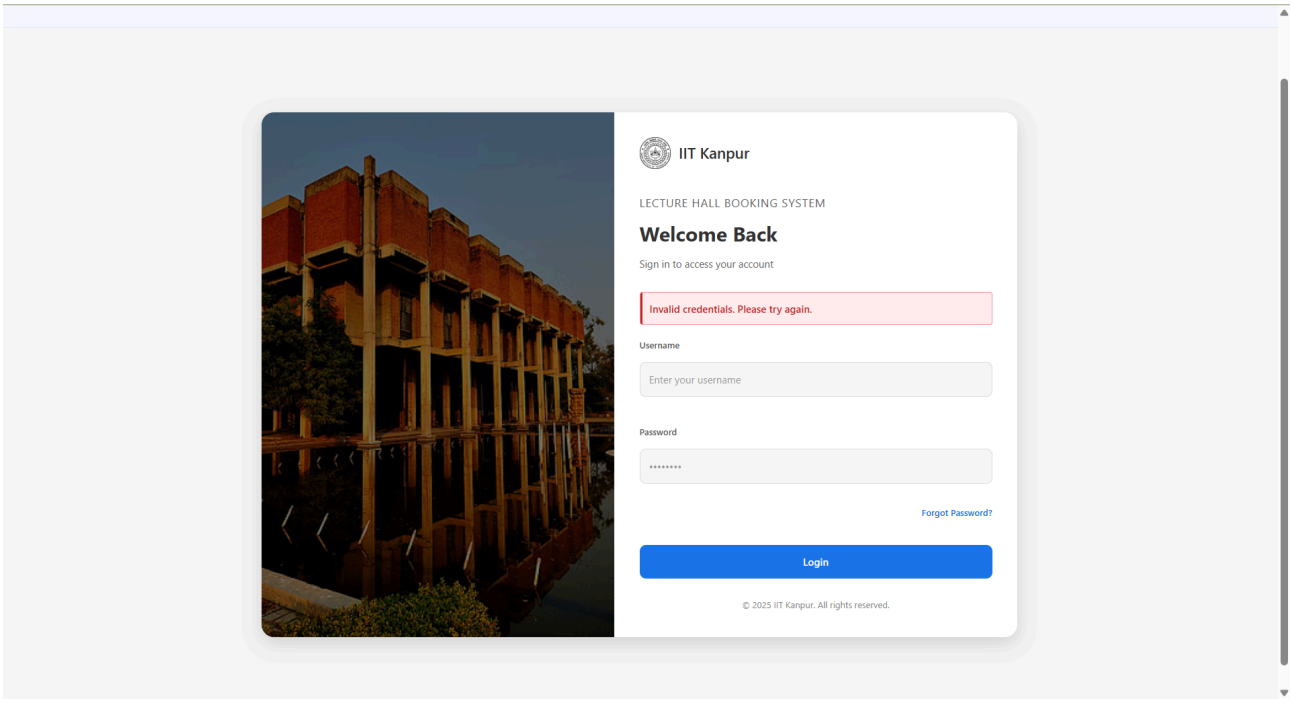
Test Results:

In this test we scrutinized the integration of the backend and frontend components of Login functionality of our web application . The backend was found to be correctly delivering with consistency.

Here we are integrating the Login api with the frontend. The integration was found to be successful, the frontend was able to send data to the database with the help of backend apis to successfully furnish the functionalities expected from Login.

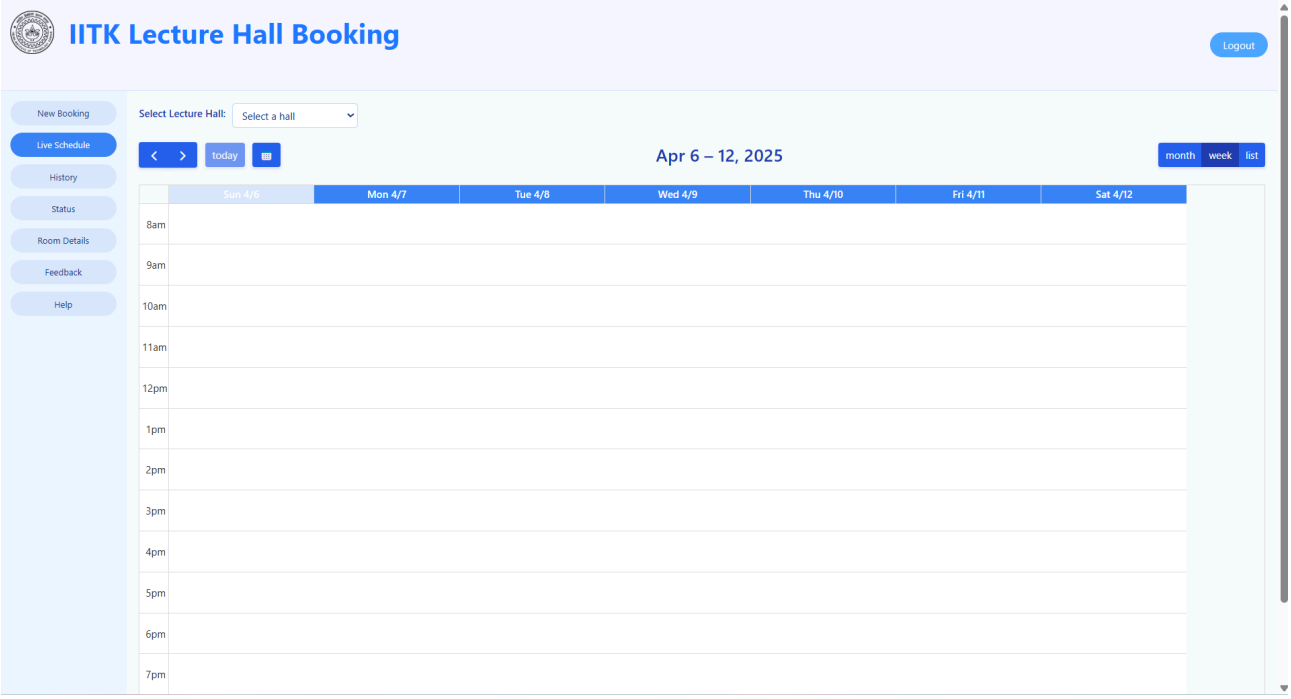
Test : Invalid Credentials(invalid user id or password)

Result : popup showing invalid credentials appears.



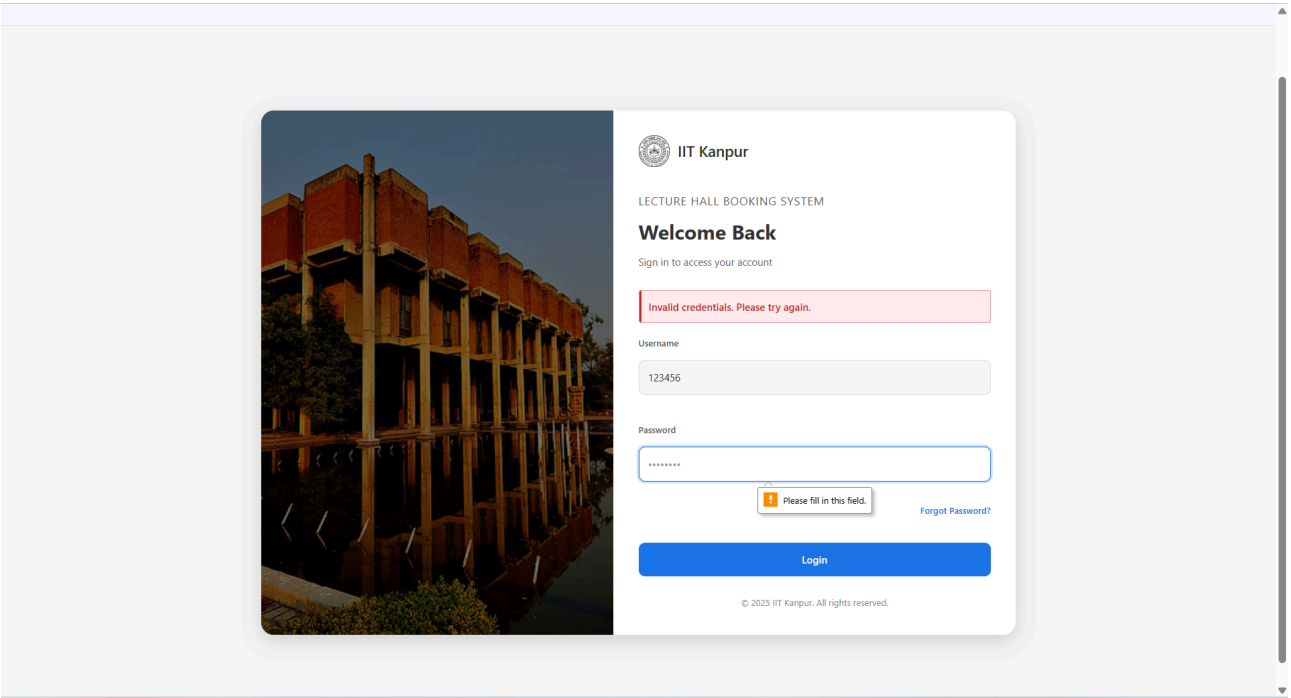
Test : Valid Credentials

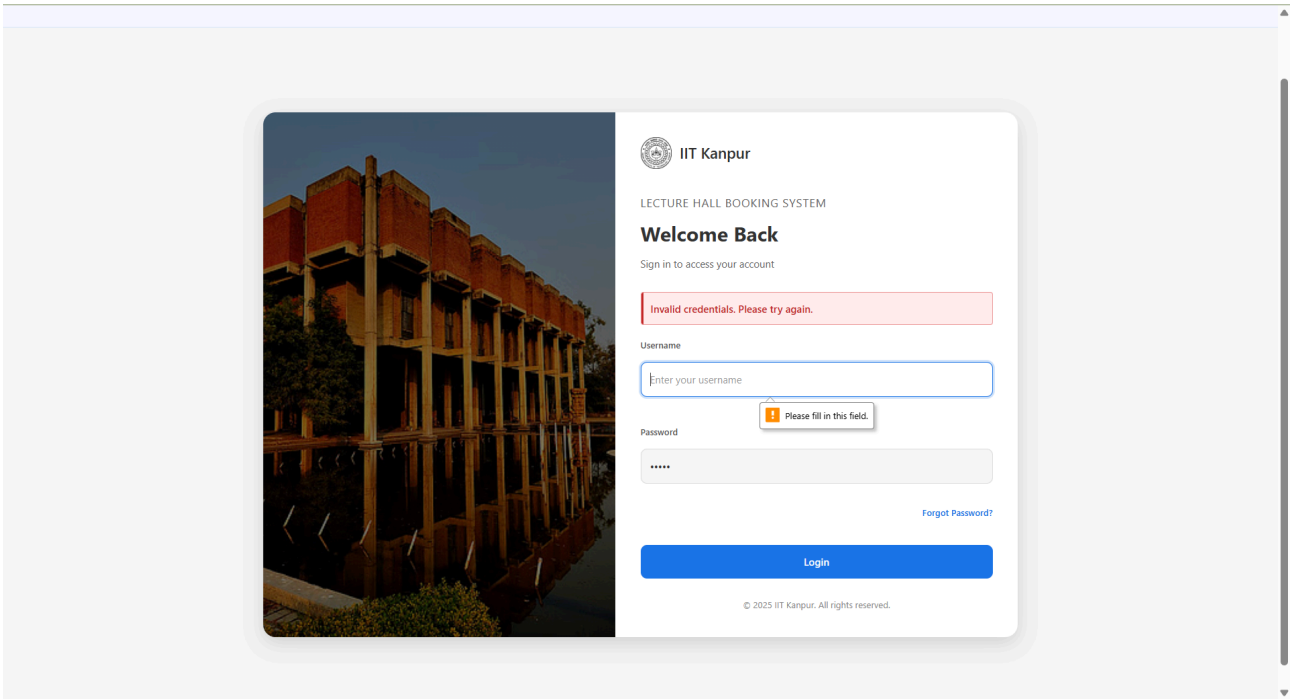
Result : Landed to Home page



Test: Blank Credentials

Result: popup showing FIll the empty field.





Additional Comments: A user is easily able to apply a booking request through the software.

2. Booking of lecture hall

Module Details: Here we tested the process for booking a lecture hall for 3 different authorities by admin ,user and faculty.

Test Owner: Aaradhya and Avantika

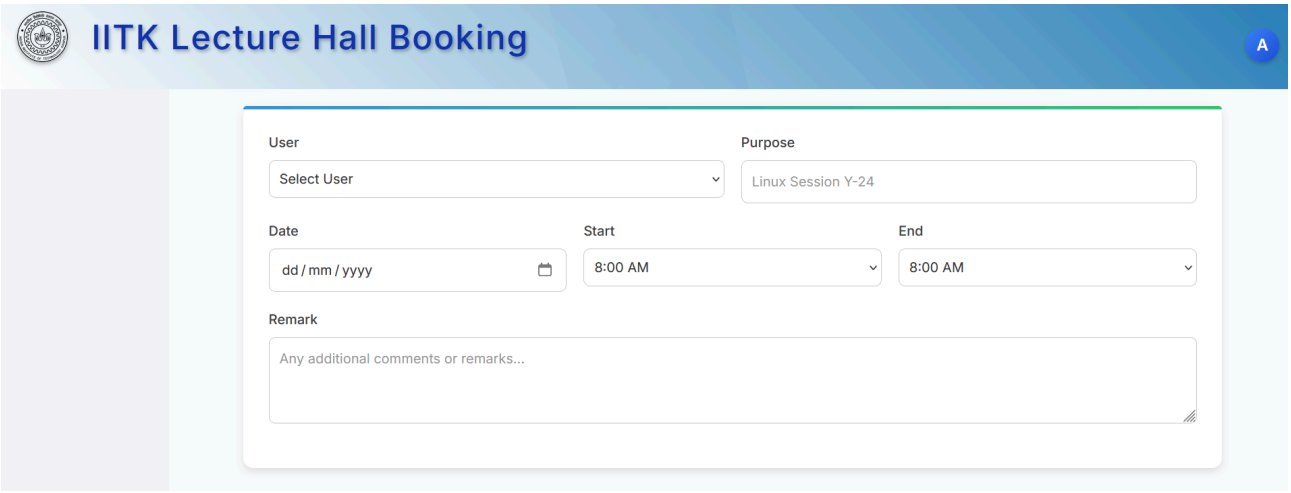
Test Date:[02/04/2025] - [02/04/2025]

Test:For admin

The admin first has to select the user for which he wants to book.

The bookings are directly confirmed

Test Results:The confirmed booking is reflected.



Test:For Faculty

The bookings are directly confirmed

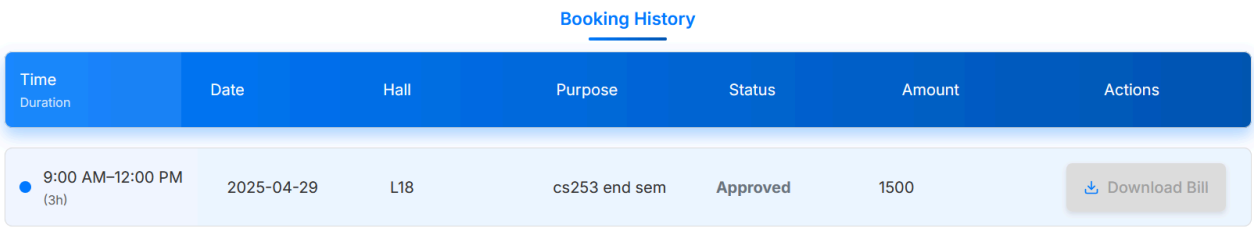
Test Results:The confirmed booking is reflected.

Test:For Student usertype

The bookings are not directly confirmed

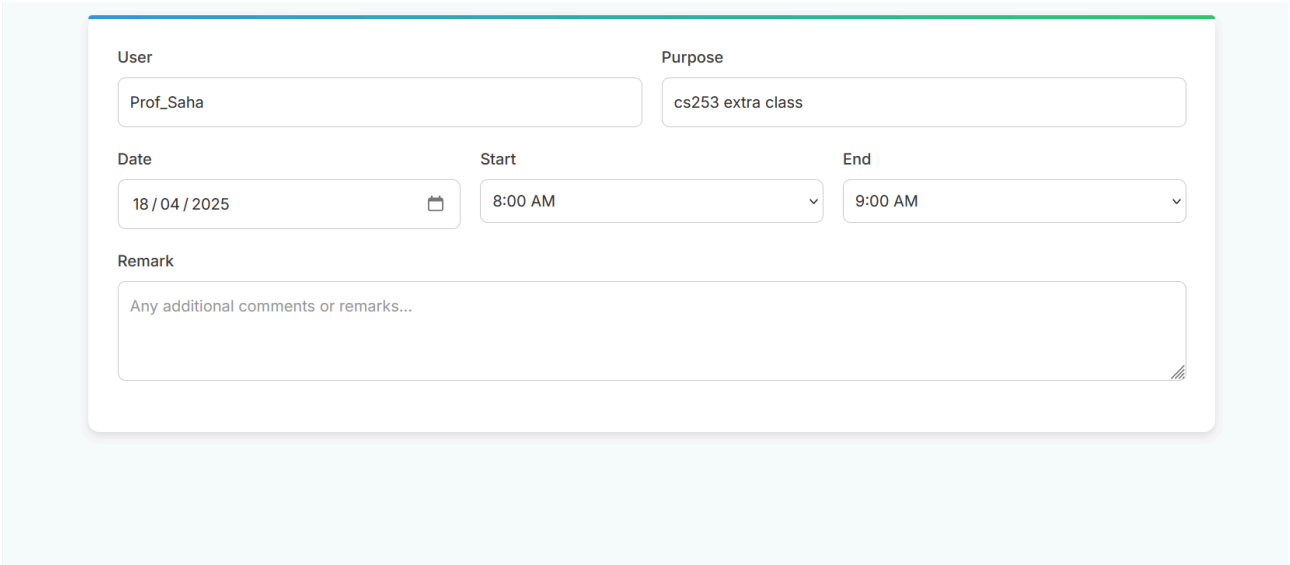
The booking needs to be verified by required authorities.

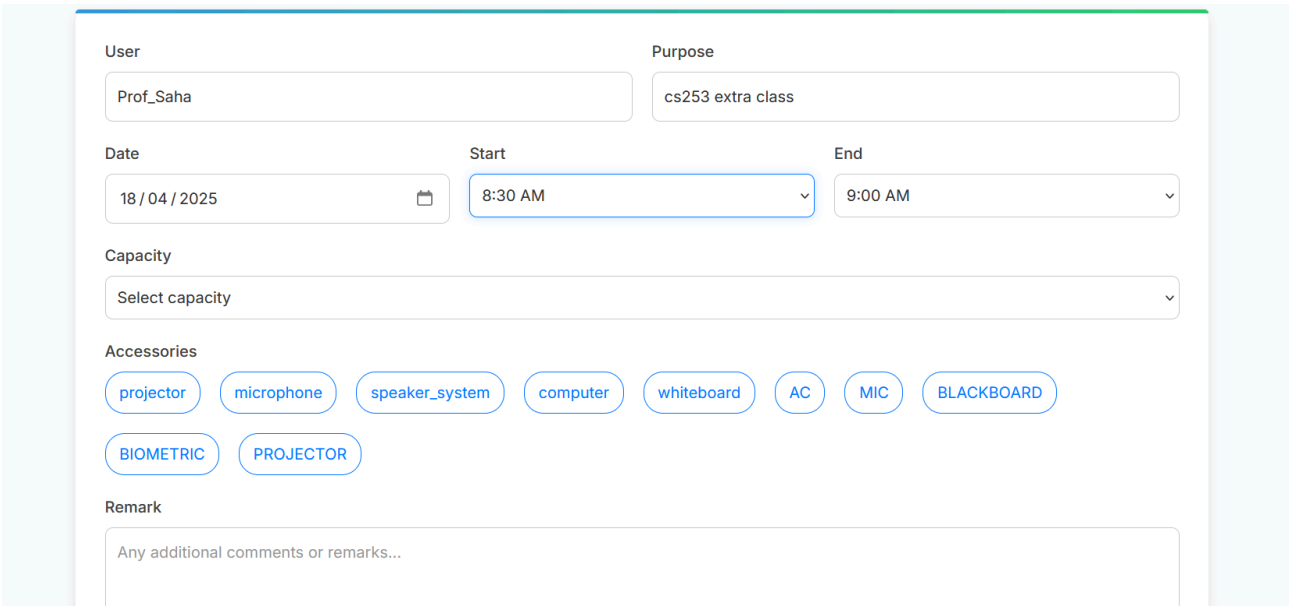
Test Results:The confirmed booking is reflected.



Test: Selecting 8:00 AM as Start Time

Test Results: A bug was encountered while selecting 8:00 AM as the start time. The system failed to register this specific time slot — clicking on the 8:00 AM option did not trigger any response, and the time was not selected. Other time slots functioned correctly. This indicates a possible issue in the time slot rendering or selection logic for this particular value.





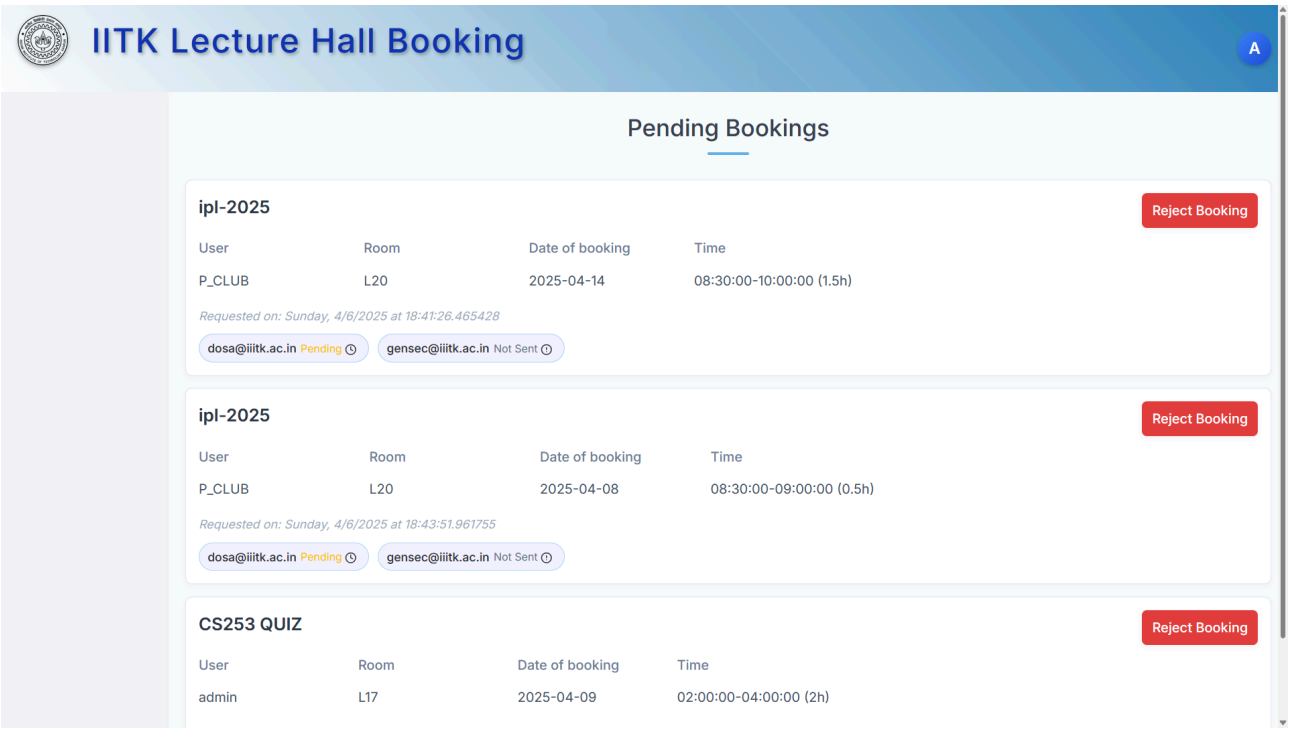
Additional Comments: A user is easily able to apply a booking request through the software.

3. Administrator's Authority to Cancel Lecture Hall Booking

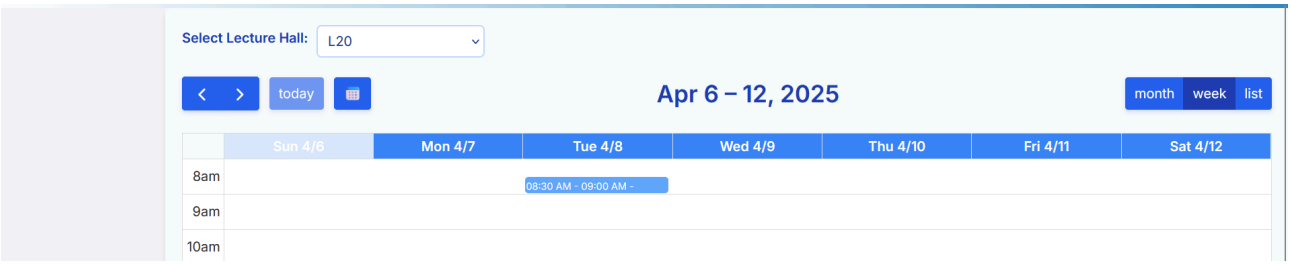
Test Owner: Bhavya and Divyesh

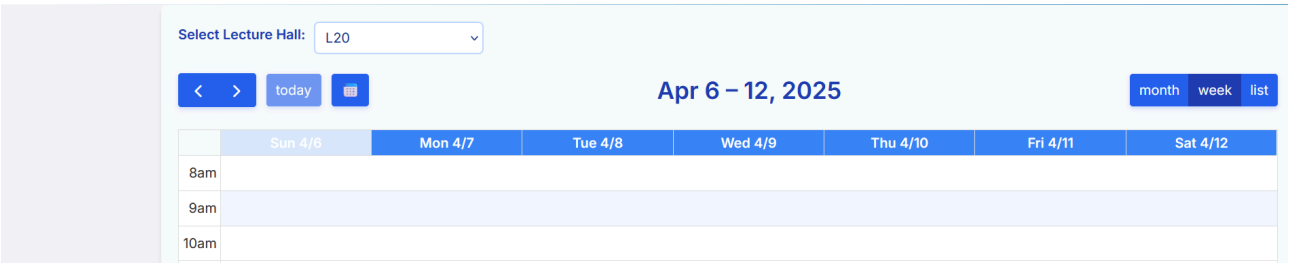
Test Date:[02/04/2025] - [02/04/2025]

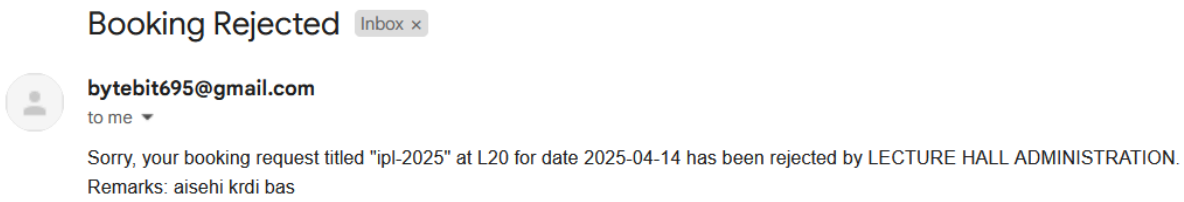
Test: Admin confirms the cancellation of a particular booking with remarks.



Test Results:The booking gets cancelled and is reflected in other parts like user and timetable.

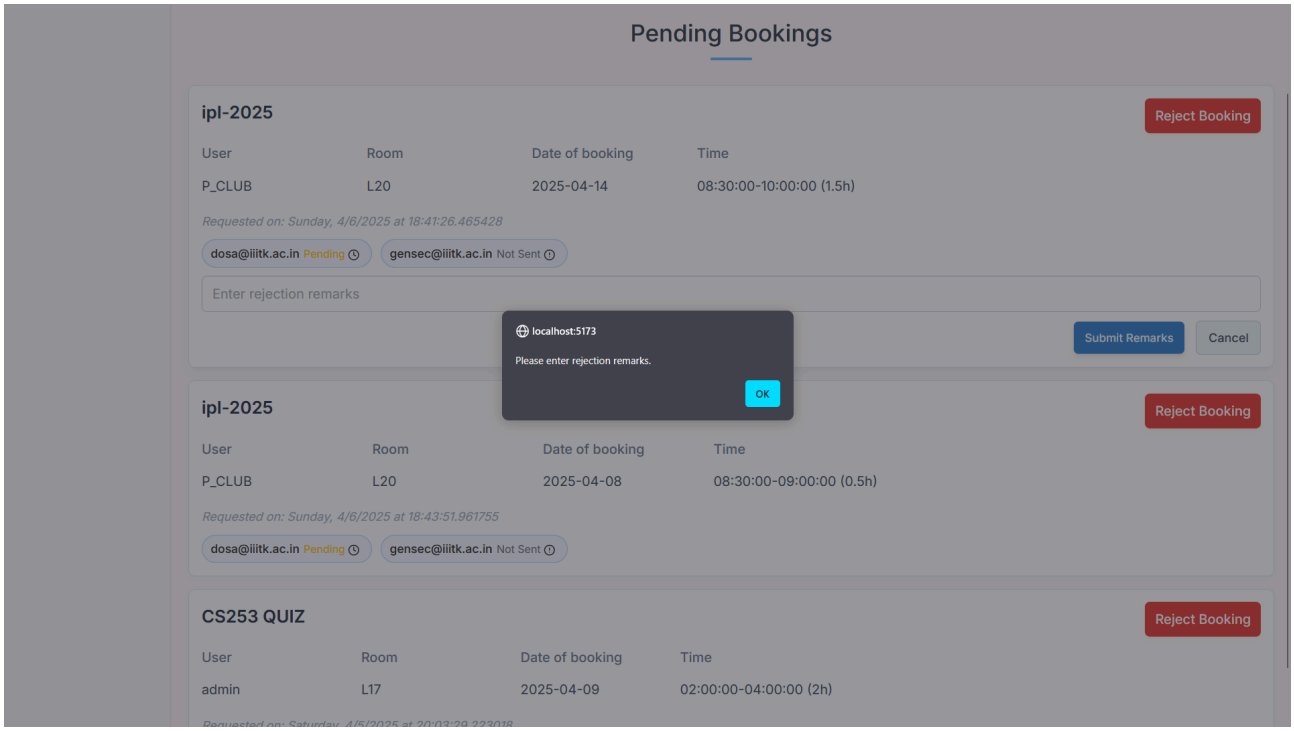






Test: Admin confirms the cancellation of a particular booking without adding any remarks.

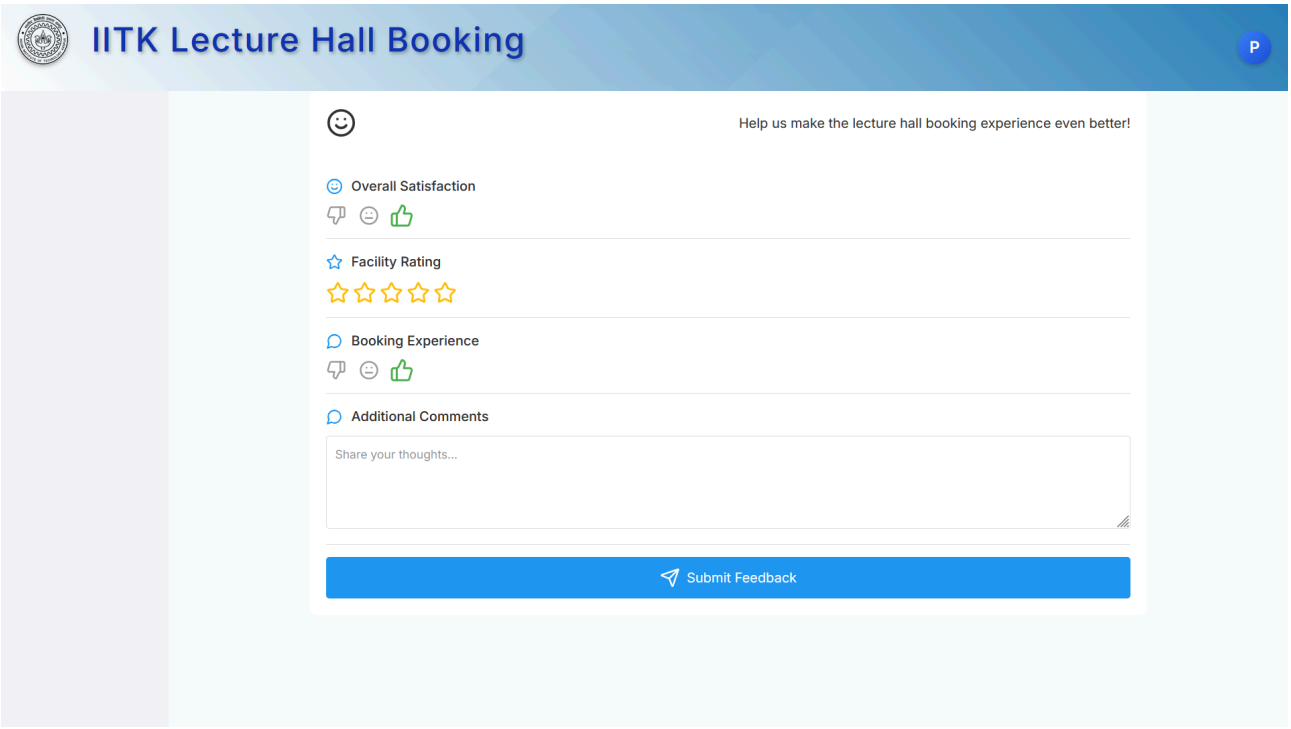
Test Results:The booking cancellation request from admin is not proceeded further due to an incomplete field.



Additional Comments: The admin has the authority to view as well as cancel any lecture hall bookings made.

4. User feedback system

Module Details: Module contains tests that verify whether users can successfully submit their feedback regarding the software and the overall lecture hall booking system.



Test Owner: Rahul and Chaitanya

Test Date:[02/04/2025] - [02/04/2025]

Test : A non-empty feedback is entered and sent.

Test Results: The feedback is successfully sent.

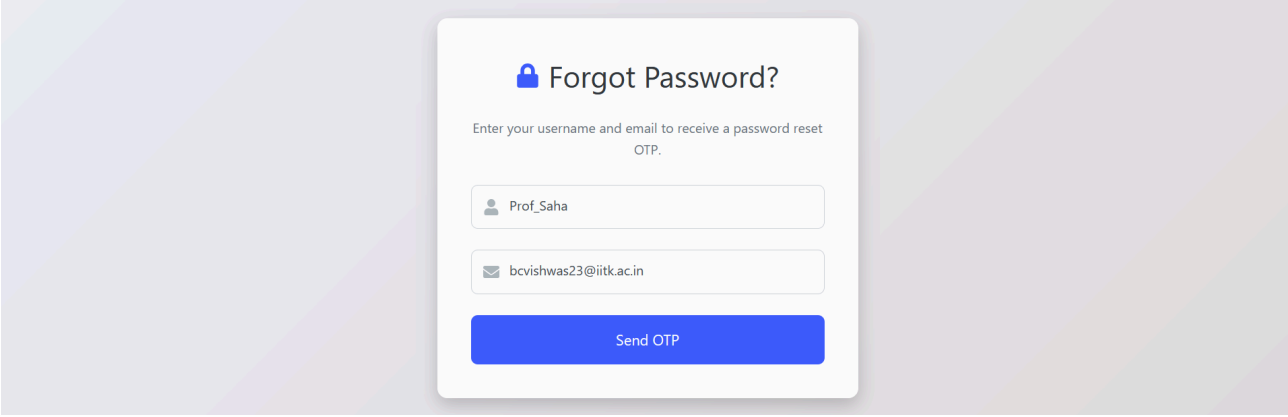
Test : An empty feedback is entered and sent.

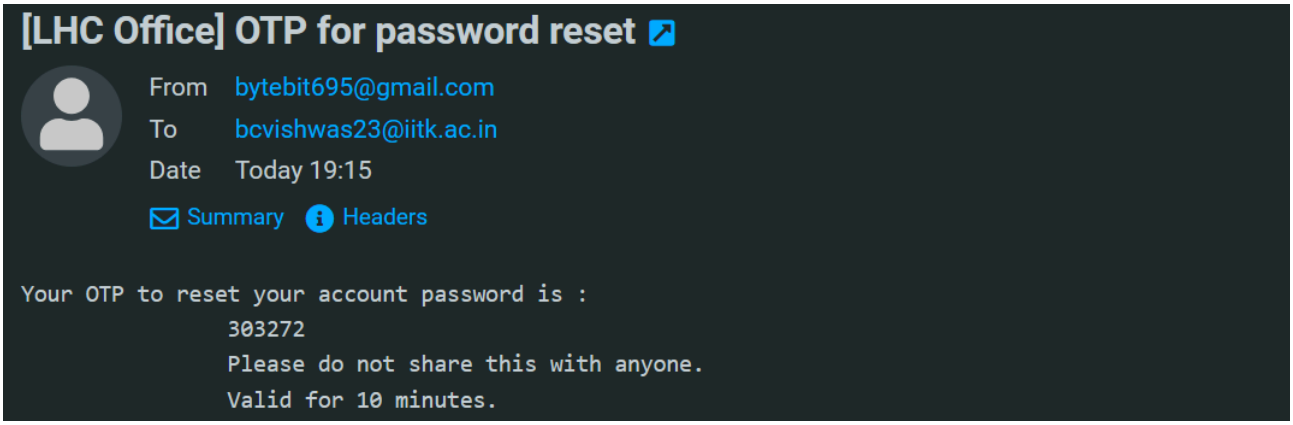
Test Results:A pop up informing that the empty fields have appeared.

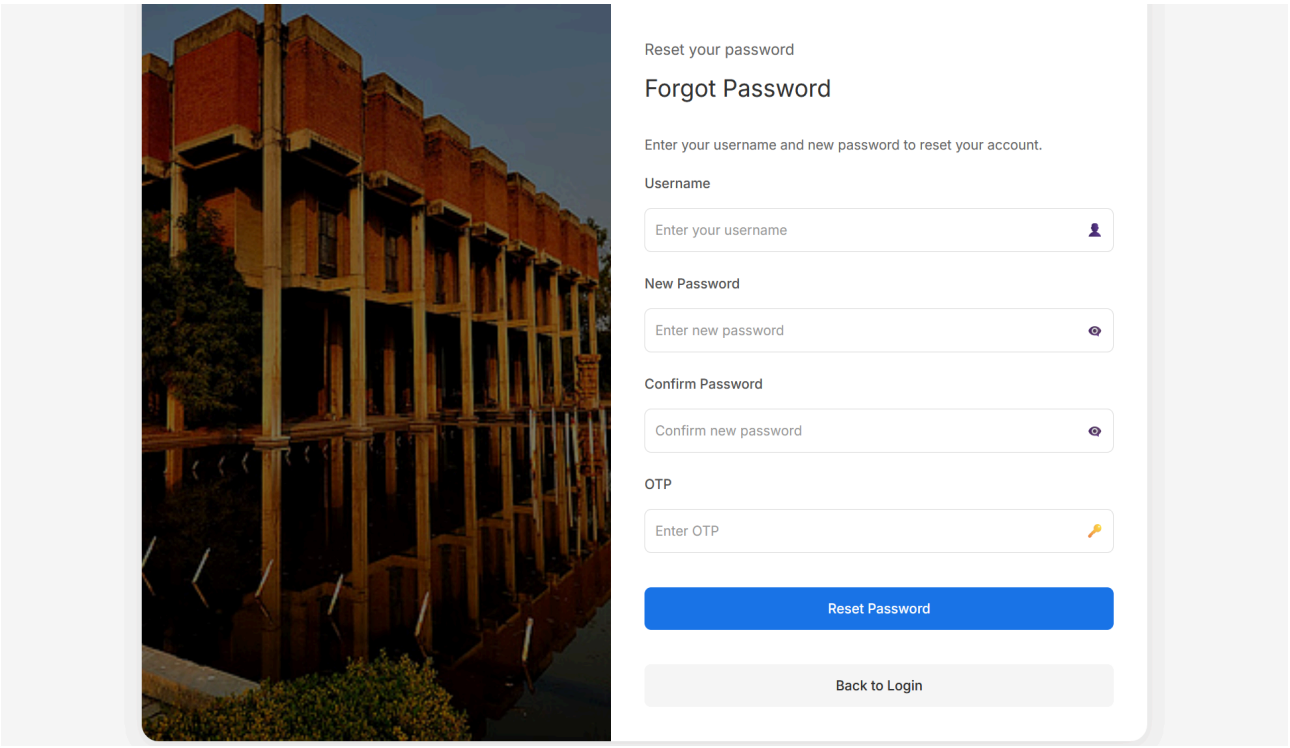
Additional Comments: The admin has the authority to view as well as cancel any lecture hall bookings made.

5. Add new admin/faculty/student

Module Details: Module tests if newly registered users(added by an admin) including new admins (no authority), students (with assigned authority), and faculty (no authority),







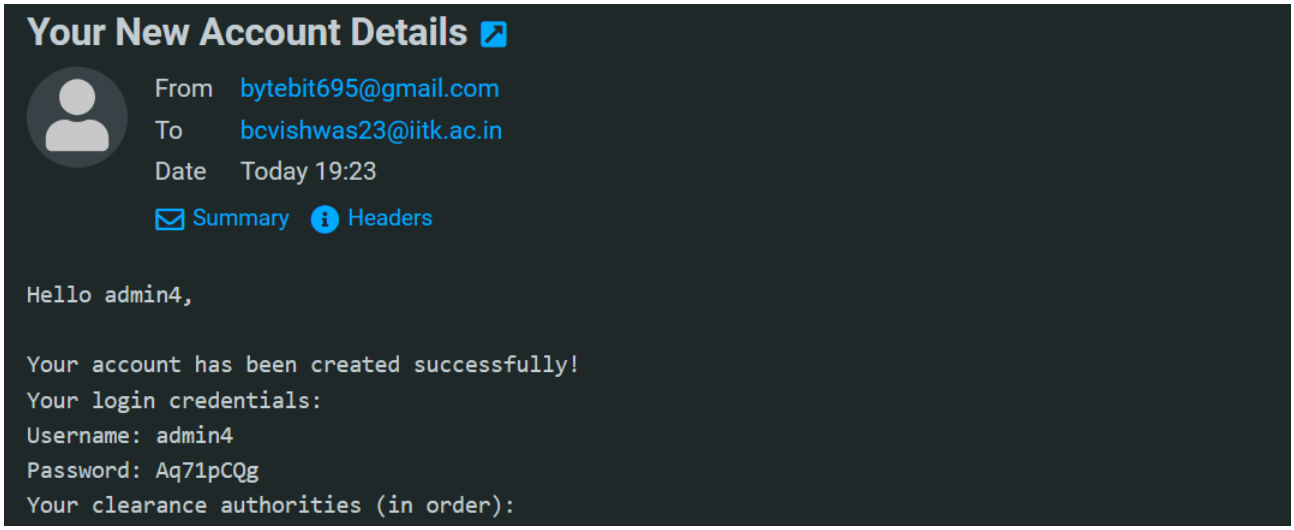
Test Owner: Atharv and Devansh

Test Date:[04/04/2025] - [04/04/2025]

Test: Creating a new admin

The admin enters the user id and email of the new admin.

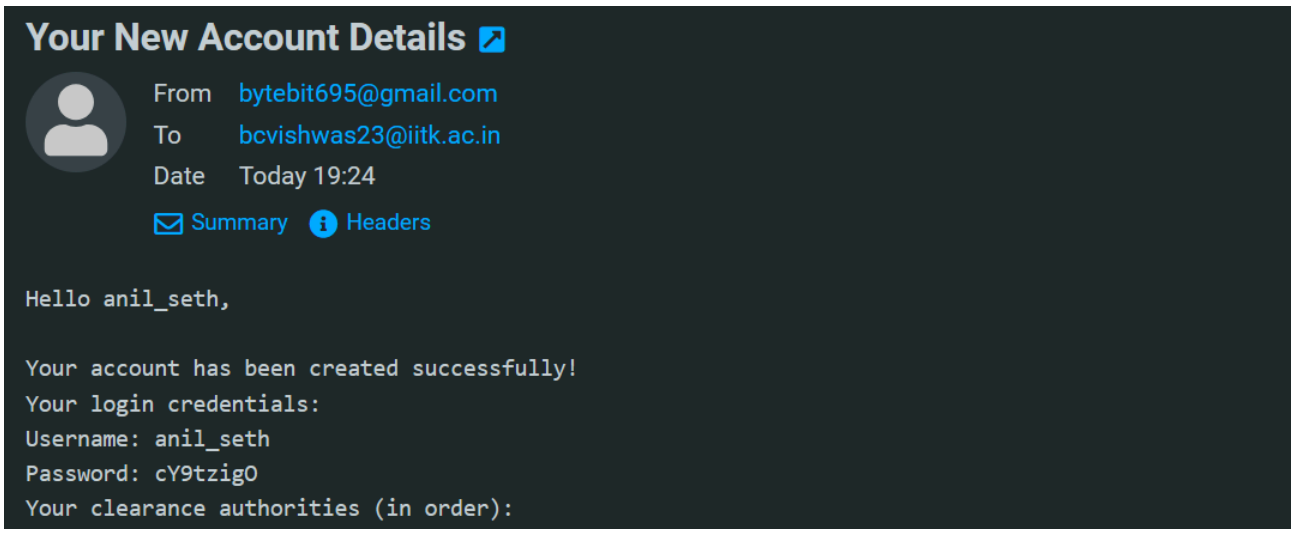
Test result: Then the new admin gets a confirmation mail with a randomly generated password.



Test: Creating a new faculty

The admin enters the user id and email of the new faculty.

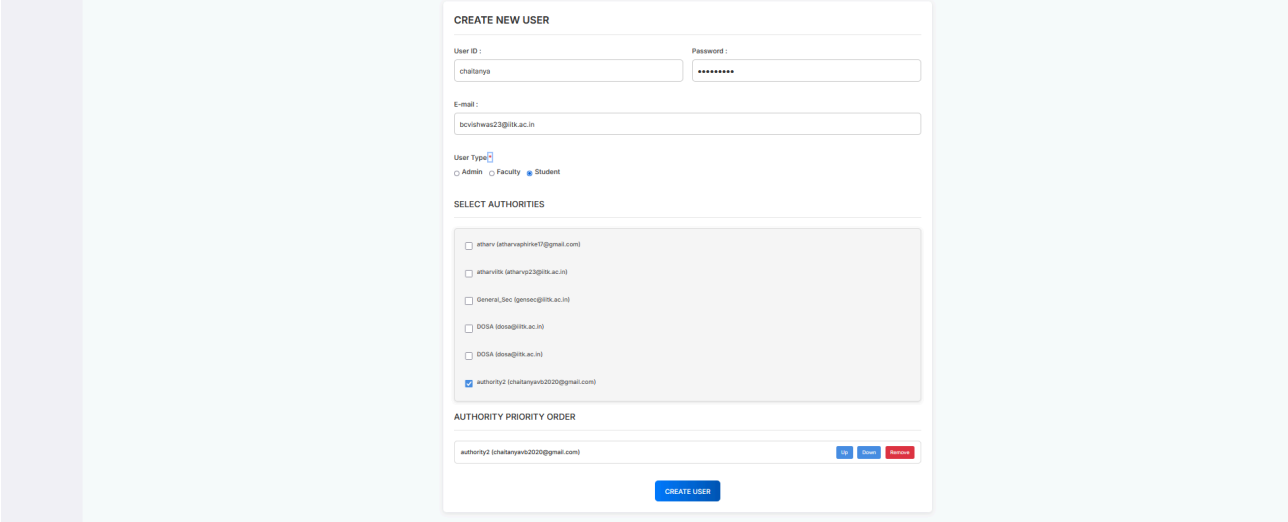
Test result: Then the new faculty gets a confirmation mail with a randomly generated password.



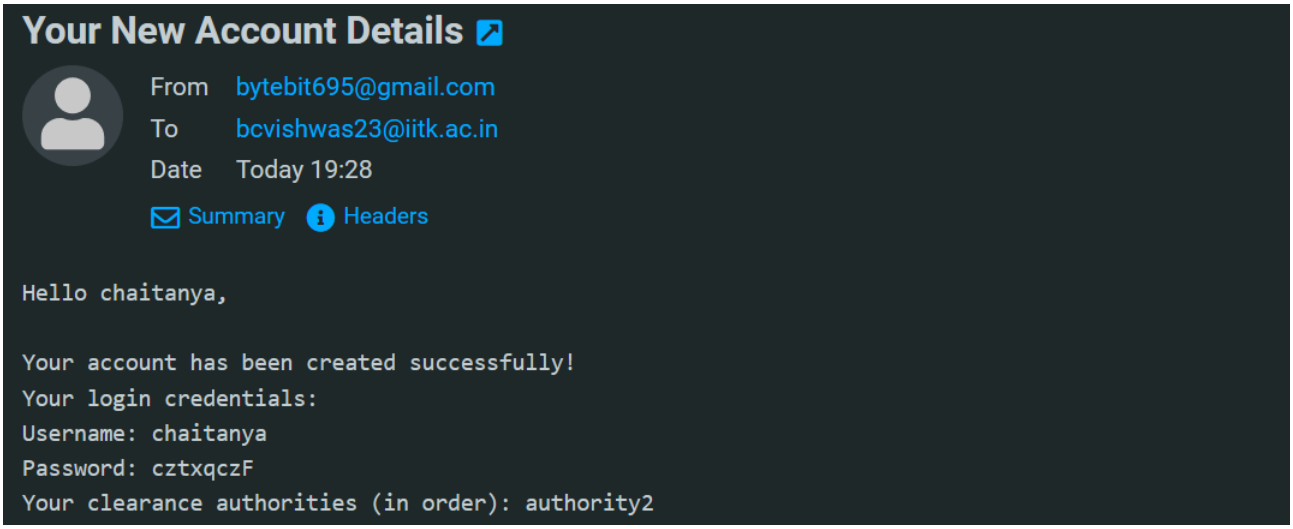
Test: Creating a new user

The admin enters the user id and email of the new user.

Then the admin should select the authorities whose approval is required for the operation

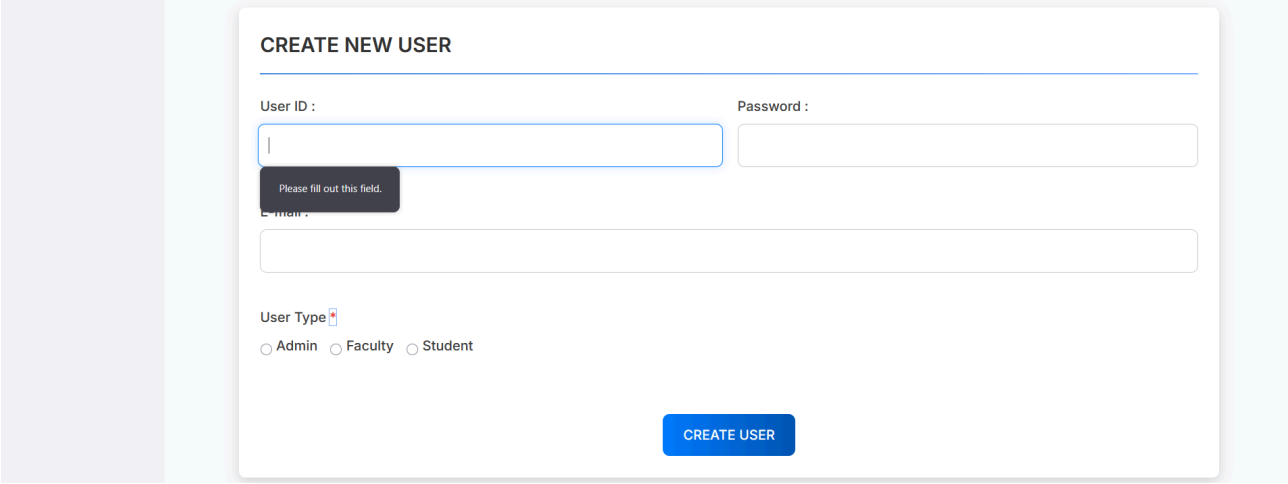


Test result: Then the new user gets a confirmation mail with a randomly generated password.



Test: The admin enters a blank field.

Test result: A pop up asking to fill the empty fields.



---

6. Forgot password

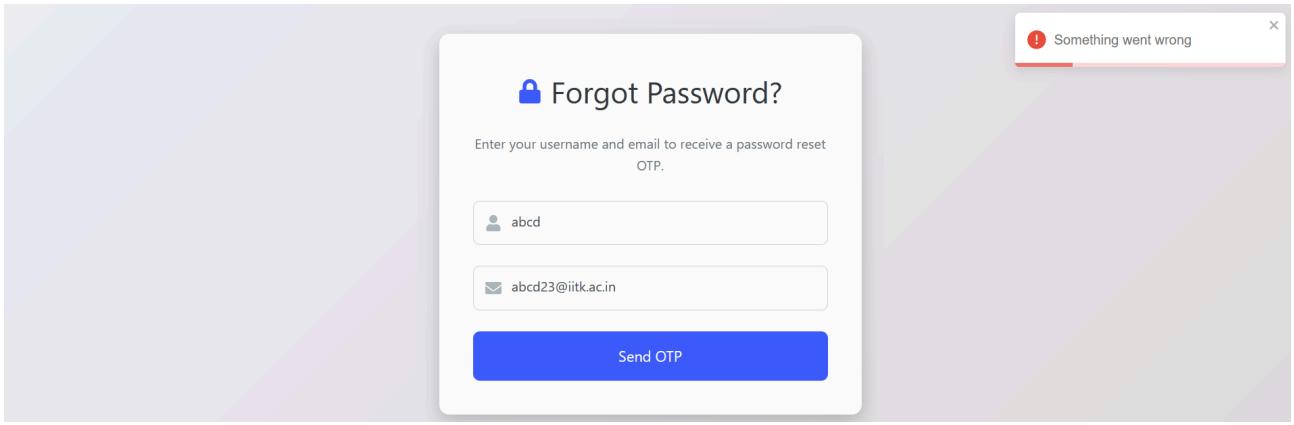
Module Details: Module contains tests that verify whether a user can successfully initiate the password reset process and receive a reset link via verification of otp sent in registered email.

Test Owner: Atharv and Devansh

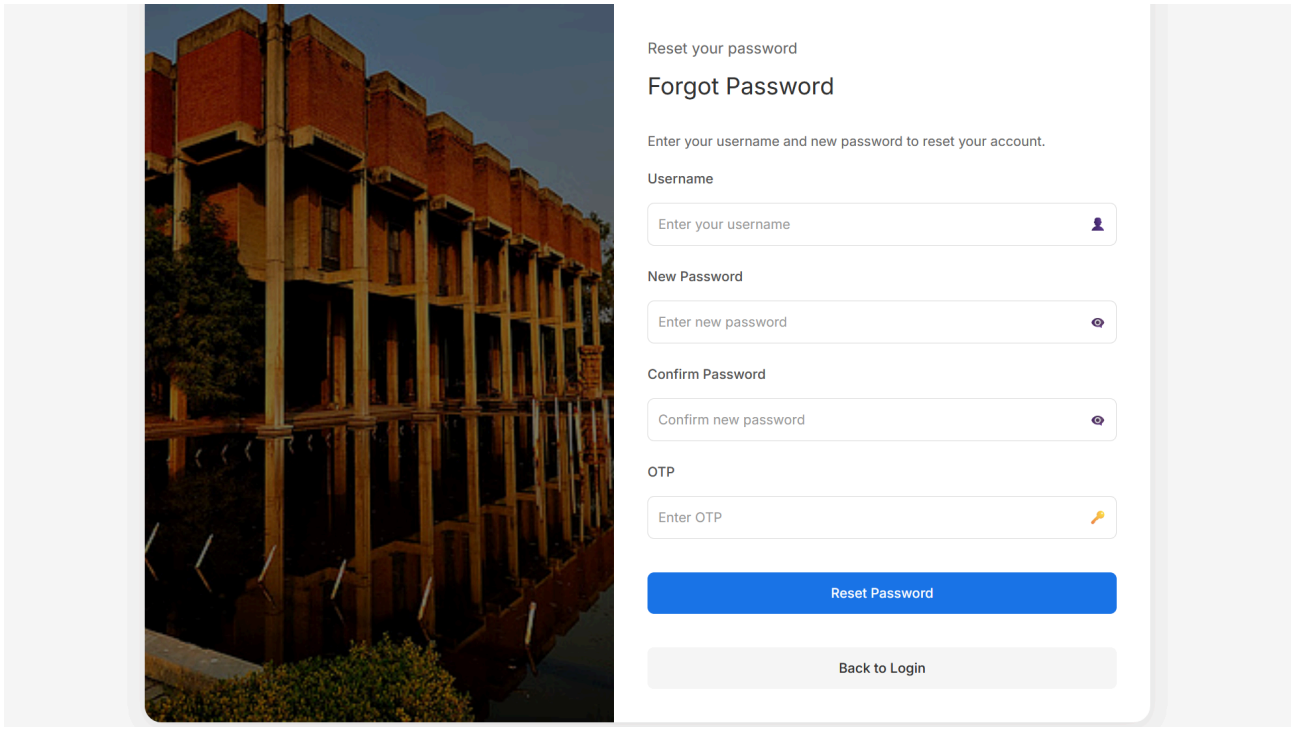
Test Date: [04/04/2025] - [04/04/2025]

Test Results:

Test: Invalid username or email id.(Username or email id are not registered).



Test: Under valid inputs(i.e.username and email), OTP is sent, hence we are redirected to the new password creation page.



7. Notifications System

Module Details:

This module verifies whether the email confirmation system is successfully integrated with booking and account modules. Confirmation emails are sent after successful actions like booking creation, password reset, and account creation.

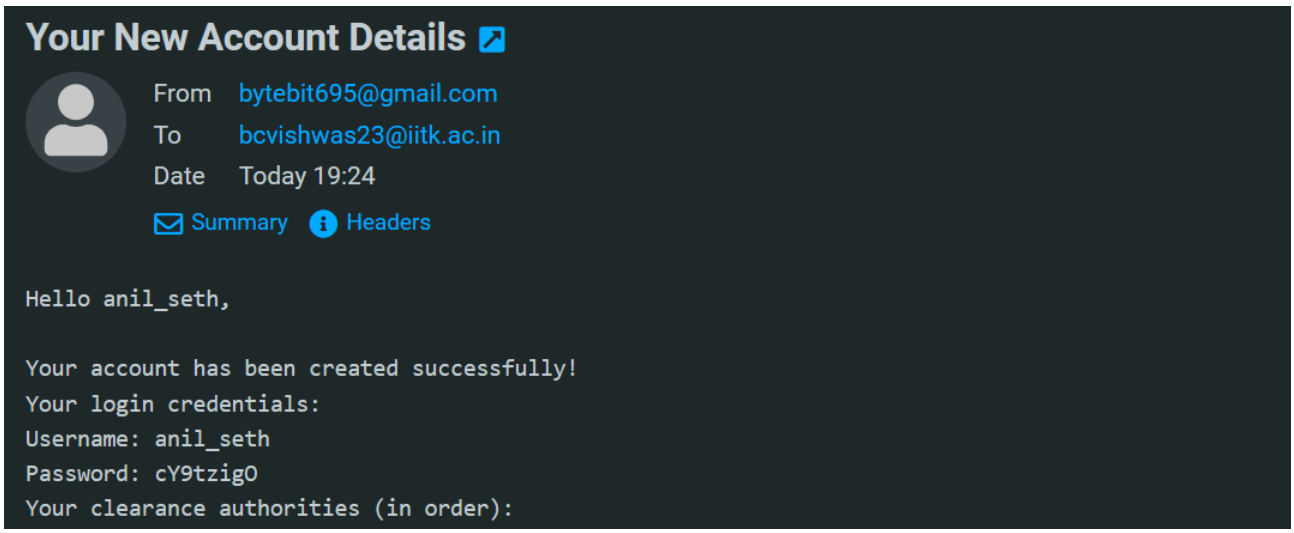
Test Owners: Aaradhya and Avantika

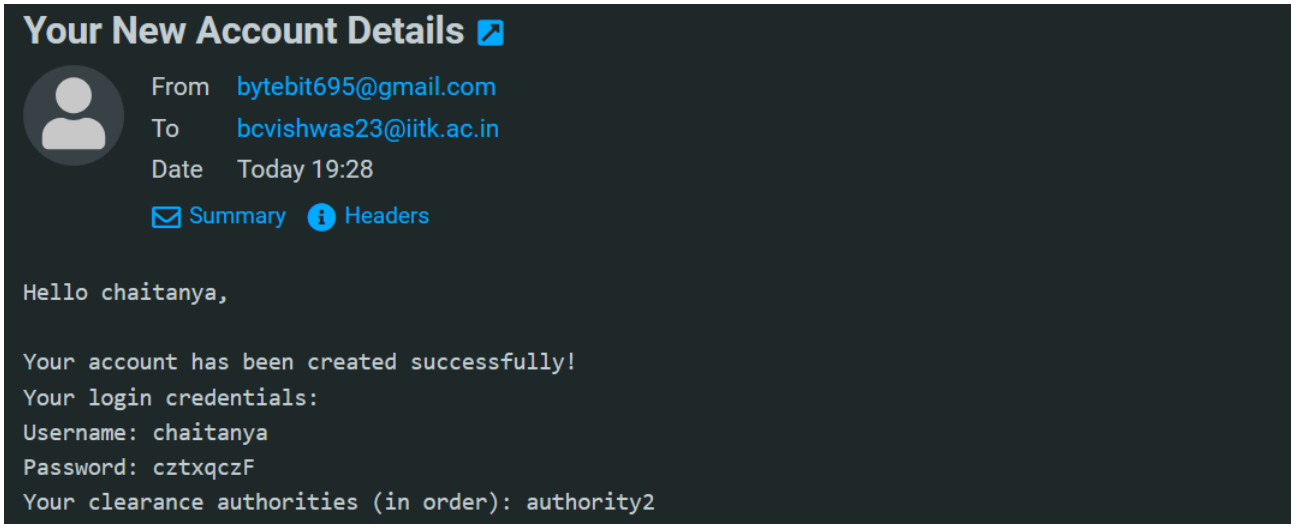
Test Date: [04/04/2025] - [04/04/2025]

Test Results:

Test: Booking Confirmation.
Input: User books a hall
Test result: Email is sent with booking details

Test: Account Creation
Input: New admin/faculty/student is added



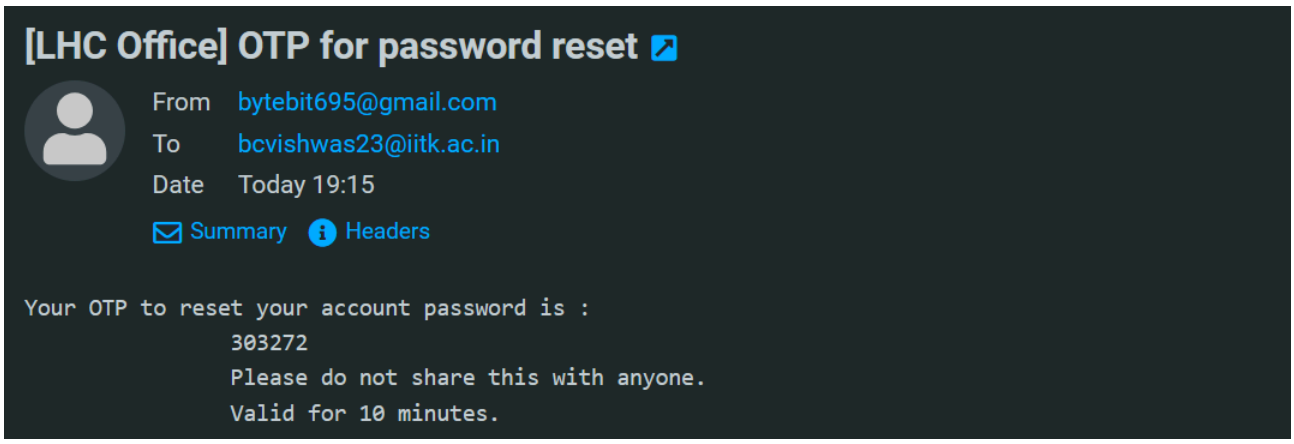


Test result: Email sent with default password

Test: Forgot Password

Input: Valid user triggers password reset

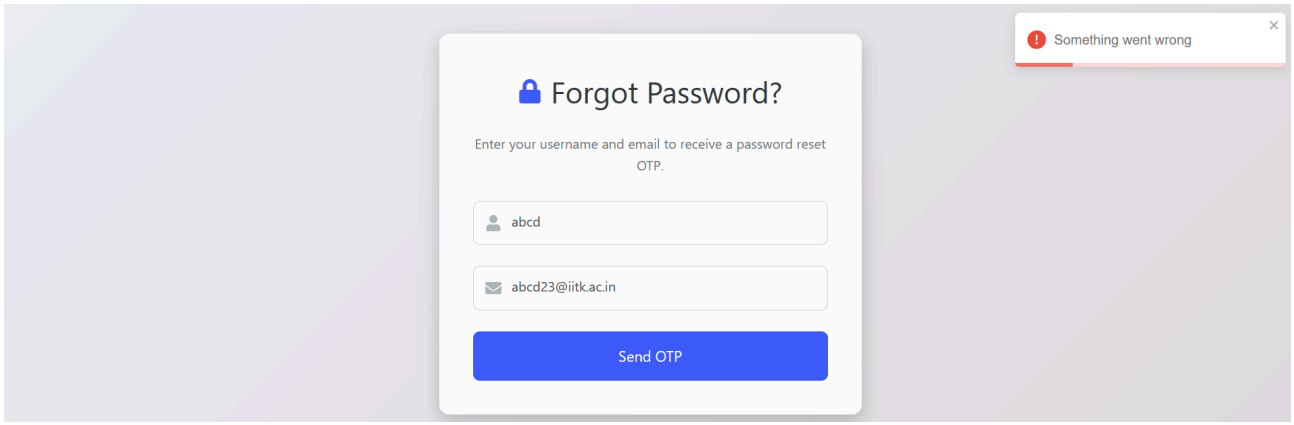
Test result: Email with OTP sent to user

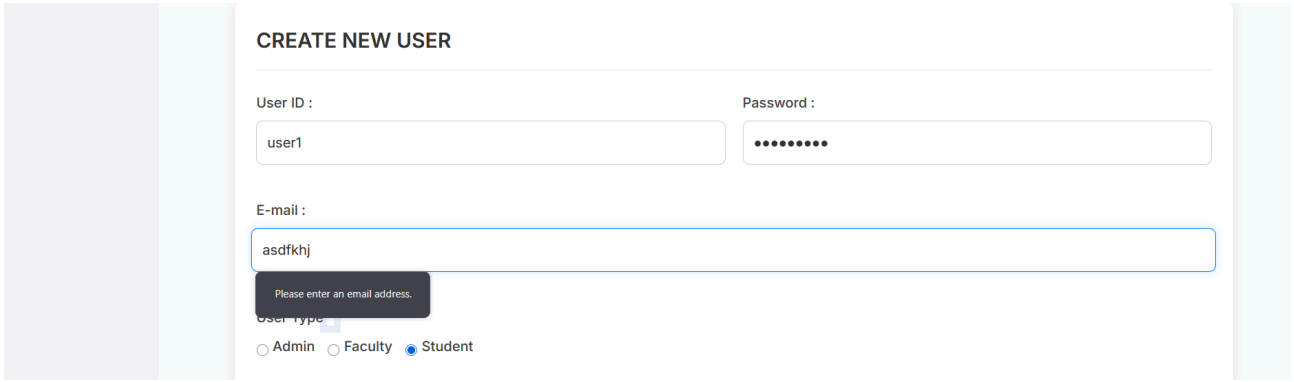


Test: Invalid Email

Input: Malformed/unregistered email

Test result: No email sent, error handled


Something went wrong
Forgot Password?
Enter your username and email to receive a password reset OTP.
abcd
abcd23@iitk.ac.in
Send OTP


CREATE NEW USER
User ID :
user1
Password :
E-mail :
asdfkhj
Please enter an email address.
Admin
Faculty
Student

4 System Testing

For system testing we used several test cases and executed them manually one by one. The test cases covered almost all the possibilities a user could conduct and we got expected results in all of them.

1. Functional Requirement: User Authentication

This involved testing both the frontend and the backend together to verify whether the user authentication system met the specified requirements. We acted as general users and attempted to log in, log out, and use the "Forgot Password" feature. All interactions were performed through the frontend, and the resulting behavior was analyzed for correctness.

For this, the hosted database was pre-populated with test user credentials. As previously mentioned, testing was conducted in parallel across multiple modules, so the user data was already available as part of other testing routines.

Test Owner: Bhavya

Test Date: [05/04/2025] -[05/04/2025]

Test Results: The tests were successful. Again, owing to rigorous unit testing, the bugs found at this stage were minimal

---

2. Functional Requirement: LH Booking (user's interface)

This involved testing both the frontend and backend to ensure the lecture hall booking system functioned as required. We simulated actions such as viewing, requesting, and canceling bookings. Students could request bookings, admins could approve or directly book on behalf of users, and faculty could book without requiring approval. The database was pre-populated with relevant data, and testing was done alongside other

modules.

Test Owner:Avantika

Test Date:[04/04/2025] - [04/04/2025]

Test Results: The tests were successful. Again, owing to rigorous unit testing, the bugs found at this stage were minimal.

---

3. Functional Requirement: Timetable

This involved testing to ensure that the timetable system functioned as expected. We simulated actions such as viewing the timetable and verifying real-time updates after bookings or cancellations. Students could view the most recent version of the timetable reflecting their requests. The database was pre-populated with scheduling data, and testing was conducted alongside other modules to ensure consistency and accuracy.

Test Owner:Areen

Test Date: [05/04/2025] -[05/04/2025]

Test Result: The tests were successful. Again, owing to rigorous unit testing, the bugs found at this stage were minimal.

---

4. Functional Requirement: Automated Invoice Generation

This involved testing to ensure accurate generation of invoices in PDF format. The system calculates the total cost based on the selected lecture hall and any additional accessories, then generates a downloadable invoice. No payment methods were implemented—only

the automated calculation and generation of the invoice were tested. Test data included predefined pricing for halls and accessories, and the functionality was verified across various booking scenarios.

Test Owner:Divyesh

Test Date: [05/04/2025] -[05/04/2025]

Test Results: The tests were successful. Again, owing to rigorous unit testing, the bugs found at this stage were minimal.

---

5. Functional Requirement: Email Services

This involved testing both the frontend and backend to ensure that the email notification system functioned correctly. The system sends emails to the appropriate authority when a student submits a lecture hall booking request. Upon approval or rejection, the user receives a corresponding confirmation email. Additionally, the "Forgot Password" feature was tested to verify that password reset emails were sent accurately to registered users. The email functionality was validated across different scenarios to ensure timely and reliable communication.

Test Owner: Atharv

Test Date: [05/04/2025] -[05/04/2025]

Test Results:The tests were successful. Again, owing to rigorous unit testing, the bugs found at this stage were minimal

---

6. **Functional Requirement: Administrative Functions**

This involved testing core administrative capabilities within the system. The administrator can cancel any user's lecture hall booking, create new user accounts, and directly add bookings without requiring any approval. These functions were tested through the admin interface to ensure proper system updates, role-based access control, and seamless interaction between frontend actions and backend logic.

Test Owner: Aaradhya

Test Date: [05/04/2025] -[05/04/2025]

Test Results:The tests were successful. Again, owing to rigorous unit testing, the bugs found at this stage were minimal.

---

7. **Functional Requirement:** Security and Data integrity

   To validate the system's security and data integrity, multiple tests were conducted:

   1. **SQL Injection Protection**:
      Performed manual SQL injection attempts through form inputs and URL parameters. Django ORM safely handled inputs and prevented malicious queries from executing.
   2. **Session Management and Timeout**:
      Confirmed that sessions expired after a predefined period of inactivity and users were automatically logged out. Also verified that Django securely stored session data.
   3. **Data Tampering**:
      Tried to modify hidden form values and intercepted requests using browser dev tools and Postman. Ensured that server-side validations were in place to prevent unauthorized data manipulation.

4. **Data Consistency Tests**:
   Simulated concurrent booking requests for the same lecture hall to ensure that no double bookings occurred.

Test Owner: Devansh

Test Date: [05/04/2025] -[05/04/2025]

Test Results: All tests passed successfully. The Django framework's built-in security features effectively prevented unauthorized access, injection attacks, and session hijacking. Data integrity was maintained across all booking and user operations, even under concurrent access.

---

8. **Non-Functional Requirement:** Responsive Web Design

To ensure that the lecture hall booking portal is accessible and visually consistent across devices, the following tests were performed:

1. Multi-Device Rendering Test:
   Verified the layout and user interface on various screen sizes including desktop (1920x1080), laptop (1366x768), tablet (768x1024), and mobile devices (360x640, 414x896). Checked layout fluidity, font scaling, button placements, and element alignment.
2. Browser Compatibility Test:
   Tested the application on multiple browsers such as Chrome, Firefox, Microsoft Edge, and Safari. Ensured that design elements and interactive components rendered uniformly.
3. Orientation Testing:
   Evaluated responsiveness by switching between portrait and landscape orientations on tablets and smartphones to ensure dynamic content adjustment.

Test Owner:Atharv

Test Date: [05/04/2025] -[05/04/2025]

Test Results: [Success]

---

9. **Non-Functional Requirement: System Response Time**

   To verify that the application responds promptly to user actions and meets acceptable response time thresholds under normal and peak conditions:

   1. Page Load Time Measurement**:**

      Measured the time taken for key pages (login, dashboard, booking form, booking history) to load completely.

   2. API Response Time Check**:**

      Timed backend API calls for operations such as login, booking creation, cancellation, and viewing availability using tools like Postman and browser network monitoring..

   3. Client-Side Performance Check**:**

      Evaluated any delays introduced by front-end rendering logic and JavaScript execution, ensuring the overall user experience remained smooth.

   4. Simulated Peak Load Response Monitoring**:**

      Simulated concurrent users (50-100) accessing the system and measured response times to ensure consistency under load.

Test Owner: Rahul

Test Date: [05/04/2025] -[05/04/2025]

Test Results: Real-time updates were successfully propagated to all connected clients. Users immediately saw changes in booking status, preventing redundant actions. No race

conditions or delayed sync issues were observed during testing.

---

11. Non-Functional Requirement: Real Time Updates

To verify the application's ability to provide real-time updates for lecture hall booking status:

1. **Simultaneous Booking Simulation:**
   Multiple users attempted to book the same lecture hall concurrently from different devices. The UI was monitored for immediate reflection of the booking status once a slot was taken.
2. **Booking Dashboard Synchronization:**
   Admin dashboard and user views were observed to ensure synchronization when bookings were added, canceled, or modified.
3. **Race Condition Handling:**
   Tested if any double bookings or inconsistent states occurred during high-traffic simulations.

Test Owner: Daksh

Test Date: [05/04/2025] -[05/04/2025]

Test Results: Real-time updates were successfully propagated to all connected clients. Users immediately saw changes in booking status, preventing redundant actions. No race conditions or delayed sync issues were observed during testing.

---

12. Non-Functional Requirement: Throughput

To assess the system's capability to handle a large number of booking requests and user actions within a short time frame:

1. **Database Stress Testing:**
   Inserted a high volume of booking records and monitored read/write performance of the PostgreSQL database under load.
2. **Response Time Monitoring:**
   Recorded the average response time under load to ensure it remained within acceptable thresholds(200 ms).
3. **Request Per Second (RPS) Measurement:**
   Anonymous user: 10 req/min
   Authenticated user: 50 req/min

Test Owner: Aaradhya

Test Date: [05/04/2025] -[05/04/2025]

Test Results: Average response time remained under 50 ms, and no data loss or timeouts were encountered. Throughput was measured at approximately 100 ms successful booking operations per second during peak.

---

5 Conclusion

How Effective and exhaustive was the testing?

Most of the unit tests had 100% decision coverage and 100% branch coverage. Integration tests covered some major features such as create booking,cancel booking,log in time table etc . Stress testing (a type of testing that is so harsh, it is expected to push the program to failure) was done at multiple concurrency levels and thread counts.

Which components have not been tested adequately?

As far as the stress testing and testing of some functional requirements were concerned we did not have adequate resources for carrying out that testing. For example, to stress test the software, we did not have adequate resources for carrying out that testing as wee required many devices and signups for them.

1. *Multiple User Registration simultaneously*
2. *Multiple Add events These are some of the inadequacies that we faced while testing.*

What difficulties have you faced during testing?

The main difficulties that we faced while testing was the process of simultaneously developing and testing the software. As we were also improving the software simultaneously, few test cases were needed to be performed again.

How could the testing process be improved?

The testing process can be improved by introducing automated testing to make the process more streamlined. Independent testers can also be used for black-box testing.

Appendix A - Group Log

Date	Timings	Duration	Minutes
25/03/25	8pm-9pm	1 hour	● Meet to discuss what to be done and written in the testing doc and user manual. ● Also revisited implementation doc to ensure everybody was on the same page.
27/03/25	6pm-9:30pm	3.5 hour	● All the team members assembled to do the unit testing together. ● All the discussion about how to proceed with the unit testing,and each of its components was done. ● Each of the parts for the unit testing was done and the unit testing part in the testing doc was edited
29/03/25	6pm-8pm	2 hours	● A meet to finish unit testing ● Also started integration testing. ● Work distribution for the user manual doc
2/04/25	7pm-9pm	2 hours	● Meet to continue the integration testing,and a few other parts of the doc like introduction etc. ● Discussed and started a small Part Of the system testing. ● Also started writing the user manual.
04/04/25	8pm-11pm	3 hours	● Continued writing user manual ● Finished integration testing and started

			system testing.
05/04/25	6pm-10pm	4 hours	Finished the user manual
06/04/25	6pm-10pm	4 hours	● Meet held to finish the testing doc. ● Also revisited and discussed all the components of the testing doc to make sure everybody was on the same page.